Office de la propriété intellectuelle du Canada

Un organisme d'Industrie Canada

Canadian Intellectual Property Office

An Agency of Industry Canada

Bureau canadien des brevets

Certification

La présente atteste que les documents ci-joints, dont la liste figure ci-dessous, sont des copies authentiques des documents déposés au Bureau des brevets.

Canadian Patent Office

Certification

This is to certify that the documents attached hereto and identified below are true copies of the documents on file in the Patent Office.

Specificationand Drawings, as originally filed, with Application for Patent Serial No: **CA 2457988**, on February 18, 2004, by **VOICEAGE CORPORATION,** assignee of Bruno Bessette, Stéphane Ragot and Joachim Thiemann, for "Methods and Devices for Audio Compression Based on Acelp/TCX Coding and Multi-Rate Lattice Vector Quantization".

CERTIFIED COPY OF PRIORITY DOCUMENT

L. Lachance

Agent certificateur/Certifying Officer

October 25, 2006

Date

Canadä

(CIPO 68)
31-03-04

OPIC CIPO

CA 02457988 2004-02-18 Industry Canada Industrie Canada

2004/02/18
051 - 04
CIPO OPIC C000200291

PETITION FOR GRANT

1. The applicant, **VOICEAGE CORPORATION**, whose complete address is 750 chemin Lucerne, suite 250, Montréal, Québec, H3R 2H6, Canada, requests the grant of a patent for an invention, entitled **"METHODS AND DEVICES FOR AUDIO COMPRESSION BASED ON ACELP/TCX CODING AND MULTI-RATE LATTICE VECTOR QUANTIZATION"**, which is described and claimed in the accompanying specification.

~~2. This application is a division of application number *, filed in Canada on *.~~

~~3. (a) The applicant is the sole inventor.~~

(b) The inventors and their complete addresses are as follows:

1. **Bruno BESSETTE**, 1546 Perodeau, Rock Forest, Québec, J1N 1L2, Canada;
2. **Stéphane RAGOT**, 10, rue Jean Rameau, 72220 Ecommoy, France; and
3. **Joachim THIEMANN**, 4657, avenue du Parc, Montréal, Québec, H2V 4E4, Canada;

and the applicant owns in Canada the whole interest in the invention.

~~4. The applicant requests priority in respect of the application on the basis of the following previously regularly filed application:~~

Country of filing	Application number	Filing date

5. The applicant appoints BROUILLETTE KOSIE PRINCE, whose complete address in Canada is 1100 René-Lévesque Boulevard West, 25th Floor, Montreal, Quebec, H3B 5C9, as the applicant's representative in Canada, pursuant to section 29 of the *Patent Act*.

6. The applicant appoints BROUILLETTE KOSIE PRINCE, whose complete address in Canada is 1100 René-Lévesque Boulevard West, 25th Floor, Montreal, Quebec, H3B 5C9, as the applicant's patent agent.

~~7. The applicant believes that the applicant is entitled to claim status as a "small entity" as defined under section 2 of the *Patent Rules*.~~

TABLE OF CONTENT

BACKGROUND OF THE INVENTION

1. Field of the Invention

The present invention relates generally to encoding and decoding of sound signals in digital transmission and storage systems, and more specifically to hybrid transform and code-excited linear prediction coding.

2. Brief Description of the Prior Art

The digital representation of information offers many well-known advantages. In the case of audio signals, the information (e.g. a speech or music signal) is usually digitized using the PCM (Pulse Code Modulation) format. The signal is thus sampled and quantized with usually 16 or 20 bits per sample. Although simple, the PCM representation results in a high bit rate (in number of bits per second or bit/s). This limitation is the main motivation for designing efficient source coding techniques which can reduce the source bit rate and meet the specific constraints of an application in terms of audio quality, coding delay, and complexity.

An audio encoder converts a sound signal into a digital bitstream which is transmitted over a communication channel or stored in a storage medium. We consider here only lossy source coding (i.e. signal compression). The role of the encoder is then to represent the PCM samples with a smaller number of bits while maintaining a good subjective audio quality. The decoder or synthesizer operates on the transmitted or stored bit stream and converts it back to a sound signal. The reader is referred to (Jayant, 1984) and (Gersho, 1992) for an introduction to signal compression methods, to the general chapters of (Kleijn, 1995) for an in-depth coverage of modern speech and audio coding techniques.

In the state of the art of high-quality audio coding, two classes of algorithms can be distinguished : *Code-Excited Linear Prediction* (CELP) coding which is designed to encode primarily speech signals, and perceptual transform (or sub-band) coding which is well adapted to represent music signals. These techniques can achieve a good compromise between subjective quality and bit rate. CELP coding has been developed in the context of low-delay bi-directional applications such as telephony or conferencing, where the audio signal is typically sampled at 8 or 16 kHz. On the other hand, perceptual transform coding has been applied mostly to wideband high-fidelity music signals sampled at 32, 44.1 or 48 kHz for streaming or storage applications.

CELP coding (Atal, 1985) is the core framework of most modern speech coding standards. In this coding model, the speech signal is processed in successive blocks of *N* samples called *frames*, where *N* is a predetermined number of samples corresponding typically to 10-30 ms. The reduction of bit rate is achieved by removing the temporal correlation between successive speech samples through linear prediction and using efficient vector quantization (VQ). A linear prediction (LP) filter is computed and transmitted every frame. The computation of the LP filter typically needs a *look-ahead*, a 5-10 ms speech segment from the subsequent frame. In general, the *N*-sample frame is divided into smaller blocks called *sub-frames*, so as to apply pitch prediction. Usually the sub-frame length is usually set in the range 4-10 ms. In each sub-frame, an excitation signal is usually obtained from two components, a portion of the past excitation and the innovative (or fixed-codebook) excitation. The component formed from the past excitation is often referred to as the adaptive codebook or pitch excitation. The parameters characterizing the excitation signal are coded and transmitted to the decoder, where the reconstructed excitation signal is used as the input of the LP filter. An important instance of CELP coding is the ACELP (*Algebraic* CELP) coding model, whereby the innovative codebook consists of interleaved signed pulses.

The CELP model has been developed in the context of narrow-band speech coding, for which the input bandwidth is 300-3400 Hz. In the case of wideband speech signals defined in the 50-7000 Hz band, the CELP model is usually used in a split-band approach, where a lower band is represented by waveform matching (CELP coding) and the higher-band is parametrically encoded. This band splitting has several motivations. Most of the bits can be allocated in a frame to the lower-band signal to maximize quality; the computational complexity (of filtering, etc.) can be reduced compared to a full-band encoding; also, waveform matching is not very efficient for high-frequency components. This split-band approach is used for instance in the ETSI AMR-WB wideband speech coding standard. This coding standard is specified in (3GPP TS 26.190) and described in (Bessette, 2002). The implementation of AMR-WB is given in (3GPP TS 26.173). The AMR-WB speech coding algorithm consists essentially of splitting the input wideband signal into a lower band (0—6400 Hz) and a higher band (6400-7000 Hz), applying the ACELP algorithm only the lower band and encoding the higher band by bandwidth extension (BWE).

The state-of-the-art audio coding techniques, e.g. MPEG-AAC or ITU-T G.722.1, are built upon perceptual transform (or sub-band) coding. In transform coding, the time-domain audio signal is processed by overlapping windows of appropriate length. The reduction of bit rate is achieved by the de-correlation and energy compaction property of a specific transform, as well as encoding only the perceptually relevant transform coefficients. The windowed signal is usually decomposed (analyzed) by a DFT, DCT or MDCT. A frame length of 40-60 ms is normally needed to achieve good audio quality. However, to represent transients and avoid time spreading of coding noise before attacks (*pre-echo*), shorter frames of 5-10 ms are also used to describe non-stationary audio segments. Quantization noise shaping is achieved by normalizing the transform coefficients by scale factors prior to quantization. The normalized coefficients are typically encoded by scalar quantization followed by Huffman coding. In parallel, a perceptual masking curve is computed to control the quantization process and optimize the subjective quality: this curve is used to encode the most perceptually relevant transform coefficients.

To improve the coding efficiency (in particular a low bit rates), band splitting can also be used with transform coding. This approach is used for instance in the new High Efficiency MPEG-AAC standard (also known as aacPlus). In aacPlus, the signal is split into two sub-bands, the lower-band signal is encoded by perceptual transform coding (AAC), while the higher-band signal is described by so-called Spectral Band Replication (SBR) which is a kind of bandwidth extension (BWE).

In applications, such as audio/video conferencing, multimedia storage and Internet audio streaming, the audio signal consists typically of speech, music and mixed content. As a consequence, it is desirable in such applications to employ an audio coding technique robust to the type of input signal. In other words, the audio coding algorithm should achieve a good and *consistent* quality for a wide class of audio signals, including speech and music. Nonetheless, the CELP technique is known to be intrinsically speech-optimized and has problems with music signals. State-of-the art perceptual transform coding on the other hand has good performance for music signals, but is not appropriate for representing speech signals, especially *at low bit rates.*

Several approaches have then been considered to encode general audio signals (including both speech and music) with a good and fairly constant quality. Transform predictive coding (Moreau, 1992), (Lefebvre,1994), (Chen, 1996), (Chen,1997) provides in particular a good foundation for the inclusion of both speech and music coding techniques into a single framework. This approach combines linear prediction and transform coding. We will consider hereafter only the technique of (Lefebvre, 1994), called TCX (Transform Coded eXcitation) coding, which is equivalent to (Moreau, 1992), (Chen, 1996) and (Chen,1997).

Originally, two variants of TCX coding have been designed (Lefebvre, 1994): one for speech signals using short frames and pitch prediction, another for music signals with long frames and no pitch prediction. In both cases, the processing involved in TCX coding can be decomposed in two steps:

1) The current frame of audio signal is processed by temporal filtering to obtain a so-called *target signal*, and then

2) The target signal is encoded in transform domain.

Transform coding of the target signal uses a DFT with rectangular windowing. Yet, to reduce blocking artifacts at frame boundaries, a windowing with small overlap has been used in (Jbira, 1998) before the DFT. In (Ramprashad, 2001), a Modified Discrete Cosine Transform (MDCT) with windowing switching is used instead ; the MDCT has the advantage to provide a better frequency resolution than the DFT while being a maximally-decimated filter-bank. However, in the case of (Ramprashad, 2001), the encoder does not operate in closed-loop, in particular for pitch analysis. In this respect, the encoder of (Ramprashad, 2001) can not be qualified as a variant of TCX.

The representation of the target signal plays a crucial role in TCX coding and controls an essential part of the TCX audio quality, because it consumes most of the available bits in every coding frame. We restrict ourselves here to transform coding in the DFT domain. Several methods have been proposed to encode the target signal in this domain, see for instance (Lefebvre, 1994), (Xie, 1996), (Jbira,1998) (Schnitzler, 1999) and (Bessette, 1999). All these methods implement a form of a gain-shape quantization, meaning that the spectrum of the target signal is first normalized by a factor (or *global gain*) g prior to the actual encoding. In (Lefebvre, 1994), (Xie, 1996) and (Jbira, 1998), this factor g is set to the r.m.s (root mean square) of the spectrum. However, in general, it can be optimized in each frame by testing different values of g, as in (Schnitzler, 1999) and (Bessette, 1999). Note that the actual optimisation of g in (Bessette, 1999) has not been disclosed. To improve the quality of TCX coding, noise fill-in (i.e. the injection of comfort noise in lieu of unquantized coefficients) has been used in (Schnitzler, 1999) and (Bessette, 1999).

As explained in (Lefebvre, 1994), TCX coding can quite successful encode wideband signals (i.e. signals sampled at 16 kHz): the audio quality is good for speech at 16 kbit/s and for music at 24 kbit/s. Yet, TCX coding is not as efficient as ACELP coding for encoding speech signals. For this reason, a switched ACELP/TCX coding strategy has been presented briefly in (Bessette, 1999). The principle of ACELP/TCX coding is similar for instance to the ATCELP (Adaptive Transform and CELP) technique of (Combescure, 1999). Obviously, the audio quality can be maximized by switching between different modes, which are actually specialized to encode a certain type of signal. For instance, CELP coding is specialized for speech and transform coding is more adapted to music, so it is natural to combine these two techniques into a multimode framework so that each audio frame can be encoded adaptively with the most appropriate coding tool. In ATCELP coding, the switching between CELP and transform coding is not seamless (i.e. it requires transition modes) ; furthermore, an open-loop mode decision is applied, i.e. the mode decision is made prior to encoding based on the available audio signal. On the contrary, ACELP/TCX has the advantage of using two *homogeneous* linear predictive modes (ACELP and TCX coding), which makes switching easier ; moreover, the mode decision is closed-loop, meaning that all coding modes are tested and the best synthesis is selected.

Note that the ACELP/TCX mode decision used in (Bessette, 1999) has never been disclosed in the prior art. Furthermore, the quantization of the TCX target signal in ACELP/TCX coding has not been disclosed into details in (Bessette, 1999). The underlying quantization method is only known to be based on *self-scalable multi-rate lattice vector quantization*, which was introduced in (Xie, 1996).

The reader is referred to (Gibson, 1988) (Gersho, 1992) for an introduction to lattice vector quantization. An *N*-dimensional lattice is a regular array of points in the *N*-dimensional (Euclidean) space. For instance, (Xie, 1996) uses an 8-dimensional lattice, known as the Gosset lattice, which is defined as:

$$RE_8=2D_8\cup\{2D_8+(1,\cdots,1)\} \qquad \text{(Eq. 1)}$$

where

$$D_8=\{(x_1,\cdots,x_8)\in Z^8 | x_1+\cdots+x_8 \textit{ is odd}\} \qquad \text{(Eq. 2)}$$

and

$$D_8+(1,\cdots,1)=\{(x_1+1,\cdots,x_8+1)\in Z^8 | (x_1,\cdots,x_8)\in D_8\} \qquad \text{(Eq. 3)}$$

This mathematical structure allows to quantize a block of 8 real numbers. RE_8 can be also defined more intuitively as the set of points (x_1, ..., x_8) verifying the properties:

i. The components x_i are signed integers (for i=1,...,8)

ii. The sum $x_1+...+x_8$ is a multiple of 4

iii. The components x_i have the same parity (for i=1,...,8), i.e. they are either all even, or all odd.

An 8-dimensional quantization codebook can then be obtained by selecting a finite subset of RE_8. Usually the mean-square error is the codebook search criterion. In the technique of (Xie, 1996), 6 different codebooks, called Q_0, Q_1, ..., Q_5, are defined based on the RE_8 lattice. Each codebook, Q_n where n=0..5, comprises 2^{4n} points, which corresponds to a rate of $4n$ bits per 8-dimensional sub-vector or $n/2$ bit per sample. The spectrum of TCX target, normalized by a scaled factor g, is then quantized by splitting it into 8-dimensional sub-vectors (or sub-bands). Each of these sub-vectors is encoded into one of the codebooks Q_0, Q_1, ..., Q_5. As a consequence, the quantization of the TCX target (after normalization by the factor *g*) produces for each 8-dimensional sub-vector a codebook number *n* indicating which Q_n has been used and an index *i* identifying a specific code-vector in Q_n. This quantization process is referred to as multi-rate lattice vector quantization, for the codebooks Q_n have different rates. The TCX mode of (Bessette, 1999) follows the same principle, yet no details are provided on the computation of the normalization factor *g* nor on the multiplexing of quantization indices and codebooks numbers.

The lattice vector quantization technique of (Xie, 1996) based on RE_8 has been extended in (Ragot, 2002) to improve efficiency and reduce complexity. However, the application of the device of (Ragot, 2002) to TCX coding has never been described in the prior art.

In the device of (Ragot, 2002), an 8-dimensional vector is coded with multi-rate quantizer that employs a set of RE_8 codebooks denoted as $\{Q_0, Q_2, Q_3, ..., Q_{36}\}$. The codebook Q_1 is not defined in the set in order to improve coding efficiency. All codebooks Q_n are constructed as subsets of the same 8-dimensional RE_8 lattice, $Q_n \subset RE_8$. The bit rate of the *n*th codebook defined as bits per dimension is 4n/8, i.e. each codebook Q_n contains 2^{4n} code-vectors. The construction of the multi-rate quantizer follows the before-mentioned reference. For a given 8-dimensional input vector, the encoder of the multi-rate quantizer finds the nearest neighbor in RE_8, and outputs a codebook number *n* and an index *i* in Q_n. Coding efficiency is improved by applying an entropy coding technique for the quantization indices (i.e. codebook numbers *n* and indices *i* of the splits). In (Ragot, 2002), a codebook number *n* is coded prior to multiplexing to the bitstream with an unary code that comprises a $n - 1$ ones and a zero stop bit. The codebook number represented with the unary code is denoted by n^E. No entropy coding is employed for codebook indices *i*. The unary code and bit allocation of n^E and *i* is exemplified in Table 1.

Table 1
The number of bits required to index the codebooks.

Codebook number n_k	Unary code n_{Ek} in binary form	Number of bits for n_{Ek}	Number of bits for i_k	Number of bits per split
0	0	1	0	1
2	10	2	8	10
3	110	3	12	15
4	1110	4	16	20
5	11110	5	20	25
...	...	...	...	...

As illustrated in Table 1, one bit is required for coding the input vector when $n = 0$ and otherwise $5n$ bits are required.

Furthermore, an important practical issue in audio coding is the formatting of the bitstream and the handling of bad frames, also known as frame-erasure concealment. The bitstream is usually formatted at the coding side as successive frames (or blocks) of bits. Due to channel impairments (e.g. CRC violation, packet loss or delay, etc.), some frames may not be received correctly at the decoding side. In such a case, the decoder typically receives a flag declaring a frame erasure and the bad frame is "decoded" by extrapolation based on the past history of the decoder. A common procedure to handle bad frames in CELP decoding consists of reusing the past LP synthesis filter, and extrapolating the previous excitation.

To improve the robustness against frame losses, parameter repetition (also know as Forward Error Correction or FEC coding) may be used.

Note that the problem of frame-erasure concealment for TCX or switched ACELP/TCX coding has never been addressed in the prior art.

OBJECTIVE OF THE INVENTION

An objective of the invention is therefore to disclose methods for efficient audio coding using a switched ACELP/TCX model and lattice (algebraic) quantizers, along with a low bit-rate bandwidth extension method for encoding the higher frequencies. Methods for multiplexing the associated variable-length frames into fixed-length packets are also disclosed, along with packet loss concealment methods appropriate for the disclosed hybrid encoder.

SUMMARY OF THE INVENTION

More particularly, in accordance with the present invention, there are provided methods for :

- switching between ACELP and TCX modes in a hybrid audio coding structure, with proper windowing and filter memory updates;
- selecting optimal coding modes and frame lengths (ACELP versus TCX of different length) in a closed-loop manner;
- applying bit-rate scalable lattice codebooks, in particular an extension of the Gosset lattice in 8-dimensions, to the gain-shape split vector quantization of a signal spectrum (in TCX modes);
- reducing the complexity of said gain-shape split vector lattice quantization by applying a novel gain estimation method, which ensures that the spectrum divided by the estimated gain will require close to the bit budget for indexing the selected lattice points after quantization;

- shaping the low-frequency coding noise in a transform coding mode (such as TCX) by applying a novel signal adaptive spectrum pre-shaping algorithm, and corresponding de-shaping;

- enhancing the performance of an ACELP-type (in particular, AMR-WB) coder for large transients, by encoding the innovative codebook gain using a form of mean-removed memoryless quantization;

- encoding the high-frequency signal (in the invention, frequencies above 6400 Hz) at low bit rate using a novel bandwidth extension method;

- splitting the bits of an encoded super-frame (80-ms of length, in the invention) into several transmission packets, while managing the possible bit overflow into one or more of the transmission packets;

- improving the robustness of TCX decoding in case of missing packets, by adding proper redundancy in the transmission of the TCX gain across the transmission packets;

The objectives, advantages and other features of the present invention will become more apparent upon reading of the following, non restrictive description of a illustrative embodiment thereof, given by way of example only with reference to the accompanying drawings.

BRIEF DESCRIPTION OF THE DRAWINGS

Figure 1 gives a high-level description of the encoder in the invention.

Figure 2 gives the timing structure of the frame types in a super-frame.

Figure 3 shows the payload structure of a packet for all frame types (in the disclosed invention, a frame can be 20-ms ACELP, of 20-ms TCX, or part of a 40-ms TCX or part of an 80-ms TCX).

Figure 4 shows the windowing used for linear predictive analysis, along with the interpolation factors used at each 5-ms sub-frame depending on the mode.

Figure 5 shows the frame windowing in ACELP/TCX encoder, depending on the present frame mode and lengtht, and the past frame mode.

Figure 6 is a high-level flow chart of the encoder for the TCX modes.

Figure 6a gives an example spectrum and associated weighting function, for the spectrum pre-shaping method disclosed in the invention.

Figure 7 shows in a block diagram how algebraic encoding is used to quantize a set of coefficients (here, frequency coefficients) based on a previously described self-scalable multi-rate lattice vector quantizer using the RE_8 lattice.

Figure 8 describes the iterative global gain estimation, for the TCX encoder. The global estimation is a critical step in TCX encoding using lattice a lattice quantizer, to reduce the complexity while remaining within the bit budget for the frame.

Figure 9 illustrates the principle of global gain estimation and noise level estimation (in TCX frames).

Figure 10 is a flowchart showing the handling of the bit budget overflow is managed in TCX encoding, when calculating the lattice point indices of the splits.

Figure 11 gives a block diagram describing the encoder for the HF signal (based on bandwidth extension).

Figure 11a shows the gain matching procedure between the low and high frequency envelope (computed in Processor 11.007 of Figure 11).

Figure 12 is a high-level block diagram of the decoder (recombination of the LF signal, encoded with hybrid ACELP/TCX, and the HF signal, encoded using bandwidth extension).

Figure 13 is a high-level block diagram of the mode extrapolation device, used when missing packets occur at the decoder.

Figure 14 is a more detailed flowchart of the mode extrapolation device.

Figure 15 illustrates the principle of ACELP/TCX decoding (for the LF signal).

Figure 16 is a flowchart showing the logic in ACELP/TCX decoding, when processing the 4 packets forming an 80-ms frame.

Figure 17 is a block diagram showing the ACELP decoding principle in the invention (details of Processor 15.007 in Figure 15).

Figure 18 is a block diagram showing the ACELP decoding principle in the invention (details of Processor 15.008 in Figure 15).

Figure 19 is a block diagram of the decoder for the HF signal, based on the bandwidth extension method disclosed in the invention.

Figure 20 is non-existant and not used in the description of the invention.

Figure 21 is a block diagram of the post-processing and synthesis filterbank at the decoder side.

Figure 22 is a flow chart iluustrating the logic in TCX global gain decoding in the presence of frame erasures, using the redundancy coding disclosed in the invention.

Figure 23 is a block diagram of the LF encoder, showing how ACELP and TCX encoders are tried in competition, using a segmental SNR criterion to select the proper encoding mode for each frame in an 80-ms super-frame.

Figure 24 is a block diagram showing the pre-processing and sub-band decomposition applied at the encoder on each 80-ms super-frame.

DETAILED DESCRIPTION OF A ILLUSTRATIVE EMBODIMENT

The illustrative embodiment of the invention discloses an audio coding device extending the ACELP/TCX model of (Bessette, 1999) and using the self-scalable multirate lattice vector quantization of (Ragot, 2002).

In the sequel, we first present an overview of the encoding principle, then the details of a illustrative embodiment of the encoder and decoder are presented.

OVERVIEW OF THE ENCODER

High-level view of the encoder

A high-level description of the encoder is shown in Fig. 1. The input signal, sampled at 16 kHz or higher, is encoded in super-frames of T ms, with $T = 80$ in the illustrative embodiment. Each super-frame is pre-processed and split into two sub-bands, in a way similar to the pre-processing of AMR-WB as disclosed in the prior art. The lower-frequency (LF) and high-frequency (HF) signals are defined in the 0-6400 and 6400-F_{max} Hz bands, respectively, where F_{max} is the Nyquist frequency which depends on the sampling frequency of the input signal.

The low-frequency signal (LF signal) is encoded by multi-mode ACELP/TCX coding built upon the AMR-WB core that operates on 20-ms frames within the 80-ms super-frame. The ACELP mode only operates on 20-ms frames since it is based on the AMR-WB encoding algorithm. The TCX mode can operate on either 20, 40 or 80 ms frames within the 80-ms super-frame. In the illustrative embodiment, the three TCX frame-lengths of 20, 40, and 80 ms are used with an overlap of 2.5, 5, and 10 ms, respectively. The overlap is necessary to reduce the effect of framing in the TCX mode (as in transform coding). Figure 2 shows the timing chart of the frame types for ACELP/TCX coding of the LF signal. ACELP mode can be chosen in any of the first, second, third and fourth 20-ms frame within an 80-ms super-frame. Similarly, TCX mode can be used in any of the first, second, third and fourth 20-ms frame within an 80-ms super-frame. Additionally, the first two, or the last two, 20-frames can be grouped together to form 40-frames to be encoded in TCX mode. Finally, the whole 80-ms super-frame can be encoded in one single 80-ms TCX frame. Hence, in total, 26 different combinations of the three TCX frame types and the ACELP frame are available to code an 80-ms super-frame. The frame types to be used (ACELP or TCX and their length) in an 80-ms super-frame are determined in closed-loop, as will be disclosed further.

The high-frequency signal (HF signal in Figure 1) is encoded using a bandwidth extension approach. In bandwidth extension, an excitation-filter parametric model is used, where the filter is encoded using few bits and where the excitation is reconstructed at the decoder from the received 18LF signal excitation. In this invention, the frame types chosen for the low-frequency band (ACELP/TCX) dictate directly the frame length used for bandwidth extension in the 80-ms super-frame.

Super-frame configurations

All possible super-frame configurations are listed in Table 2 in the form (m_1, m_2, m_3, m_4) where m_k denotes the frame type selected for the kth frame of 20 ms inside the super-frame such that

$m_k = 0$ for 20-ms ACELP,

$m_k = 1$ for 20-ms TCX,

$m_k = 2$ for 40-ms TCX,

$m_k = 3$ for 80-ms TCX.

For example, the configuration (1, 0, 2, 2) indicates that the 80-ms super-frame is encoded by encoding the first 20-ms frame with20-ms TCX, followed by encoding the second 20-ms frame with 20-ms ACELP and finally by encoding the last two 20-ms frames as a single 40-ms TCX frame. Similarly, the configuration (3, 3, 3, 3) indicates that 80-ms TCX is used for the whole super-frame.

(0, 0, 0, 0)	(0, 0, 0, 1)	(2, 2, 0, 0)	
(1, 0, 0, 0)	(1, 0, 0, 1)	(2, 2, 1, 0)	
(0, 1, 0, 0)	(0, 1, 0, 1)	(2, 2, 0, 1)	
(1, 1, 0, 0)	(1, 1, 0, 1)	(2, 2, 1, 1)	
(0, 0, 1, 0)	(0, 0, 1, 1)	(0, 0, 2, 2)	
(1, 0, 1, 0)	(1, 0, 1, 1)	(1, 0, 2, 2)	
(0, 1, 1, 0)	(0, 1, 1, 1)	(0, 1, 2, 2)	(2, 2, 2, 2)
(1, 1, 1, 0)	(1, 1, 1, 1)	(1, 1, 2, 2)	(3, 3, 3, 3)

Table 2. All possible 26 super-frame configurations.

Mode selection

The super frame configuration can be determined either by open-loop or closed-loop decision. The open-loop method consists in selecting the super-frame configuration following some analysis before the super-frame encoding in a way to reduce the overall complexity. In closed-loop, the approach consists in trying all super-frame combinations and choosing the best one. A closed-loop decision generally provides higher quality compared to open-loop decisions, with a tradeoff on complexity. In the illustrative embodiment, the closed-loop decision is performed as summarized in Table 3 and explained below.

In the closed-loop mode decision, all 26 possible super-frame configurations of Table 2 can be selected with only 11 trials. The left half of Table 3 (Trials) shows what encoding mode is applied to each 20-ms frame at each of the 11 trials. Fr0 to Fr3 refer to Frame 0 to Frame 3 in the super-frame. The trial number (1 to 11) indicates a step in the closed-loop mode-selection process. The final mode decision is known only after step 11. Note that each 20-ms frame is involved in only four of the 11 encoding trials. When more than 1 frame is involved in a trial (lines 5, 10 and 11), then TCX of the corresponding length is applied (TCX40 or TCX80). To understand the intermediate steps of the mode decision process, the right half of Table 3 gives an example of mode selection, where the final decision (after trial 11) is 80-ms TCX. This would result in sending a value of 3 for the mode in all four packets for this super-frame. Bold numbers in the example at the right of Table 3 show at what point a mode decision is taken in the intermediate steps of the mode selection process.

	TRIALS (11)				**Example of selection (In bold = comparison is made)**			
	Fr 1	**Fr 2**	**Fr 3**	**Fr 4**	**Fr 1**	**Fr 2**	**Fr 3**	**Fr 4**
1	ACELP				ACELP			
2	TCX20				**ACELP**			
3		ACELP			ACELP	ACELP		
4		TCX20			ACELP	**TCX20**		
5	TCX40	TCX40			**ACELP**	**TCX20**		
6			ACELP		ACELP	TCX20	ACELP	
7			TCX20		ACELP	TCX20	**TCX20**	
8				ACELP	ACELP	TCX20	TCX20	ACELP
9				TCX20	ACELP	TCX20	TCX20	**TCX20**
10			TCX40	TCX40	ACELP	TCX20	**TCX40**	**TCX40**
11	TCX80	TCX80	TCX80	TCX80	**TCX80**	**TCX80**	**TCX80**	**TCX80**

Table 3. Trials and example of closed-loop mode selection

The mode selection process shown in Table 3 proceeds as follows. First, in trials 1 and 2, ACELP (AMR-WB) then 20-ms TCX encoding are tried on the first 20-ms frame (Fr0). Then, a mode selection is made for Fr0 between these two modes. The selection criterion in the illustrative embodiment is the segmental Signal-to-Noise Ratio (SNR) between the weighted signal and the synthesized weighted signal. The segmental SNR is computed using 5-ms segments. In the example of Table 3, we assume that mode ACELP was retained. Then, in trial 3 and 4, the same mode comparison is made for Fr1 between ACELP and 20-ms TCX. Here, we assume that 20-ms TCX was better than ACELP, again based on the segmental SNR measure disclosed above. This choice is indicated in bold on line 4 of the example at the right of Table 3. Then, in trial 5, Fr0 and Fr1 are grouped together to form a 40-ms frame which is encoded using 40-ms TCX. The algorithm now has to choose between 40-ms TCX for the first two frames, compared to ACELP in the first frame and TCX20 in the second frame. In this example, on line 5 in bold, the sequence ACELP-TCX20 was selected, according to the segmental SNR criterion.

The same procedure as trials 1 to 5 is then applied to the third and fourth frames (Fr2 and Fr3), in trials 6 to 10. After trial 10, in the example of table 3, the four 20-ms frames are classified as : ACELP for Fr0, then TCX20 for Fr1, then TCX40 for Fr2 and Fr3 grouped together. A last trial (line 11) is he performed when all four 20-ms frames (the whole super-frame) are encoded with 80-ms TCX. Using the segmental SNR criterion, again with 5-ms segments, this is compared with the signal encoded using the mode selection in trial 10. In the example of Table 3, we assume that the final mode decision is 80-ms TCX for the whole super-frame. The mode bits for each 20-ms frame would then be (3,3,3,3) as discussed in Table 2.

Overview of the TCX mode

The closed-loop mode selection disclosed above implies that the samples in a super-frame have to be encoded using ACELP and TCX before making the mode decision. ACELP encoding is performed as in AMR-WB. TCX encoding is performed as shown in the block diagram of Figure 6. The TCX encoding principle is similar for TCX frames of 20, 40 and 80 ms, with a few differences mostly involving the windowing and filter interpolation. The details of TCX encoding will be given in the detailed description of the encoder. For now, we summarize the TCX encoding of Figure 6 as follows. The input audio signal is filtered through a weighting filter (same perceptual filter as in AMR-WB) to obtain a weighted signal. The weighting filter coefficients are interpolated in a fashion which depends on the TCX frame length. If the past frame was an ACELP frame, the zero-input response (ZIR) of the weighting filter is removed from the weighted signal. The signal is then windowed (the window shape will be described in the detailed description) and a transform is applied to the windowed signal. In the transform domain, the signal is first pre-shaped, to minimize coding noise artifact in the low-frequencies, and then quantized using a specific lattice quantizer that will be disclosed in the detailed description. After quantization, the inverse pre-shaping function is applied to the spectrum which is then inverse transformed to provide a quantized time-domain signal. After gain rescaling, a window is again applied to the quantized signal to minimize the block effects of quantizing in the transform domain. Overlap-and-add is used with the previous frame if it was in also TCX mode. Finally, the excitation signal is found through inverse filtering with proper filter memory updating. This TCX excitation is in the same "domain" as the ACELP (AMR-WB) excitation.

The details of the TCX encoding principle shown in Figure 6 will be described below.

Overview of the Bandwidth extension

Bandwidth extension is a method used to encode the HF signal at low cost, in terms of bit-rate and complexity. In the illustrative embodiment, we use an excitation-filter model to encode the HF signal. The excitation is not transmitted at all; rather, the decoder extrapolated the HF signal excitation from the received, decoded LF excitation. Hence, no bits are required for the HF excitation signal. All the bits for the HF signal are used to transmit an approximation of the spectral envelope. A linear LPC model (the filter) is computed on the down-sampled HF signal of Figure 1. These LPC coefficients can be encoded with few bits. This is because the resolution of the ear decreases at higher frequencies, and the spectral dynamics of audio signals also tends to be smaller at high frequencies. A gain is also transmitted for every 20-ms frame. This gain is required to compensate for the fact that the HF excitation signal (extrapolated from the LF excitation signal) does not match the transmitted LPC filter for the HF signal. The LPC filter is quantized in the ISF domain.

Coding in the low- and high-frequency bands is time-synchronous such that bandwidth extension is segmented over the super-frame according the mode selection in the lower band. The bandwidth extension module will be disclosed in the detailed description of the encoder.

Encoding Parameters

The coding parameters can be divided into three categories as shown in Figure 1; superframe configuration information (or mode information), LF signal parameters and HF signal parameters.

The *super-frame configuration* can be coded using different approaches. In particular, to meet specific systems requirements, it is often desired to send large packets (as the 80-ms super-packets described herein) as a sequence of smaller packets, corresponding to fewer bits and possibly shorter duration. We disclose here the specific option of dividing each 80-ms super-frame into four consecutive, smaller packets. For partitioning a super-frame into four packets, it is convenient to represent the frame type chosen for each 20-ms frame inside a super-frame using two bits to be included in the corresponding packet. This can be accomplished readily by mapping an integer $m_k \in \{0, 1, 2, 3\}$ into its binary representation. Recall that m_k is an integer describing the mode selected for the *kth* 20-ms frame in a super-frame.

The low-frequency parameters depend on the frame type. In *ACELP frames*, the parameters are the same as those of AMR-WB, in addition to a mean-energy parameter used in this invention to improve the performance of AMR-WB on attacks in music signals, as disclose here. Specifically, when a 20-ms frame is encoded in ACELP mode (mode 0), the parameters sent for that frame in the corresponding packet are :

- The ISF parameters (46 bits reused from AMR-WB)
- The Mean energy (2 additional bits compared to AMR-WB)
- Pitch lag (as in AMR-WB)
- Pitch filter (as in AMR-WB)
- Fixed codebook indices (reused from AMR-WB)
- Codebook gains (as in 3GPP AMR-WB)

In *TCX frames*, ISF parameters are the same as in ACELP mode (AMR-WB), but they are transmitted only once every TCX frame. For example, if the super-frame is comprised of two 40-ms TCX frames, then only two sets of ISF parameters are transmitted for the whole super-frame. Similarly, if the super-frame is encoded as only one 80-ms TCX frame, then only one set of ISF parameters is transmitted for that super-frame. For each TCX frame (either 20ms, 40ms, 80ms) , the following parameters are transmitted :

- One set of ISF parameters (46 bits reused).

- Parameters describing the quantized spectrum coefficients in the multi-rate lattice VQ (refer to Figure 6)
- Noise factor for noise fill-in (3 bits).
- Global gain (scalar, 7 bits).

These parameters and their encoding will be disclosed in the detailed description of the encoder. Note that a large portion of the bit budget in TCX frames is dedicated to the lattice VQ indices.

The high-frequency parameters, which are provided by the *Bandwidth extension*, are typically only related to spectrum envelop and energy. The following parameters are transmitted :

- One set of ISF parameters (order 8, 9 bits) per frame (a frame can be 20-ms ACELP, or 20-ms TCX, or 40-ms TCX or 80-ms TCX)
- HF Gains (7 bits), quantized as a 4-dimensional gain vector, with one gain per 20, 40 or 80-ms frame
- Gains correction (for 40-ms TCX and 80-ms TCX only) to modify the more coarsely quantized HF gains in these TCX modes.

Bit allocations in the illustrative embodiment

The ACELP/TCX codec in this illustrative embodiment can operate at five bit rates : 13.6, 16.8, 19.2, 20.8 and 24.0 kbit/s. These bit rates are related to some of the AMR-WB rates, which is an integral part of the invention. The corresponding number of bits to encode each 80-ms super-frame at the five rates given above is 1088, 1344, 1536, 1664, and 1920 bits, respectively. In total, 8 bits are allocated for the super-frame configuration (2 bits per 20-ms frame) and 64 bits for bandwidth extension in each 80-ms super-frame. More or fewer bits could be used for the bandwidth extension, depending on the resolution desired to encode the

HF gain and spectral envelope. The remaining bit budget (i.e. most of the bit budget) is used to encode the low frequency signal (LF signal in Figure 1). As an illustration, a typical bit allocation for the different frame types is detailed in Tables 4 and 5. The bit allocation of bandwidth extension is shown in Table 6. These tables serves as an indication of the percentage of the total bit budget typically used for encoding the different parameters in the invention. Note that in tables 5b and 5c, corresponding respectively to 40-ms and 80-ms TCX, the numbers in parentheses show the splitting of the bits into two (table 5b) or 4 (table 5c) packets of equal size. For example, table 5c indicates that in TCX-80 mode, the 46 ISF bits of the super-frame (only one LPC filter for the super-frame) are split as : 16 bits in the first packet, then 6 bits in the second packet, then 12 bits in the third packet and finally 12 bits in the last packet. Similarly, the algebraic VQ bits (most of the bit budget in TCX modes) are split into two packets (table 5b) or four packets (table 5c). This splitting is done in such a way that the quantized spectrum is split into two (table 5b) or four (table 5c) interleaved tracks, where each track contains one out of every two (table 5b) or one out of every four (table 5c) spectral block. Each spectral block is composed of 4 successive complex spectrum coefficient. This interleaving ensures that, if a packet is missing, it will only cause interleaved "holes" in the decoded spectrum for 40-ms TCX and 80-ms TCX. This splitting of bits into smaller packets for 40-ms TCX and 80-ms TCX has to be done carefully, to manage overflow when writing into a given packet.

DETAILED DESCRIPTION OF THE ENCODER

In the illustrative embodiment, the audio signal is assumed to be sampled in the PCM format at 16 kHz or higher, with a resolution of 16 bits per sample. The role of the encoder is to compute and encode some parameters based on this signal, and to transmit the encoded parameters into the bitstream for decoding and synthesis purposes. A flag indicates to the coder what is the input sampling rate.

The input signal is divided into successive blocks of 80 ms, which will be referred to as *super-frames* hereafter. A simplified block diagram of the encoder is shown in Figure 1. Each super-frame is pre-processed, and then split into two sub-bands (Processor 1.001) in a way similar to AMR-WB speech coding. The lower-frequency (LF) and high-frequency (HF) signals are defined in the [0-6400] and [6400-11025] Hz bands, respectively.

As was disclosed in the encoder overview, the low-frequency (LF) signal is encoded by multimode ACELP/TCX coding (Processor 1.002), while the high-frequency (HF) signal is coded by HF extension (Processor 1.003). The coding parameters computed in a given 80-ms super-frame (i.e. the mode information, as well as the quantized HF and LF parameters) are multiplexed into 4 packets of equal size.

In what follows, the main blocks of the diagram of Figure 1 (pre-processing and analysis filter-bank, LF encoding and HF encoding) are discussed in their respective details.

Pre-processing and analysis filterbank

Figure 24 shows a block diagram of the pre-processing and sub-band decomposition in the illustrative embodiment of the encoder. The input signal (a super-frame of 80-ms duration) is first separated in two sub-signals in Processors 24.001 and 24.002. The sub-signals are respectively the Low-Frequency (LF) and High-Frequency (HF) signals in the output of Processor 1.001. Hence, Processor 24.001 performs downsampling with proper filtering to obtain the HF signal, and Processor 24.002 performs downsampling with proper filtering to obtain the LF signal, in a method similar to AMR-WB sub-band decomposition. The HF signal will then be the input of the high-frequency coding module (Processor 1.003 in Figure 1). The LP signal is further pre-processed by two filters before being passed to the LF signal encoding module (Processor 1.002 of Figure 1) : first, the LF signal is filter with a high-pass filter having cutoff frequency 50 Hz (Processor 24.003) – this is to remove the DC component and the very low frequency components ; then, after high-pass filtering, the LF signal is filter with a deemphasis filter (Processor 24.004) to accentuate the high-frequency components. This pre-emphasis is typical in wideband speech coders.

LF encoding

A simplified block diagram of the LF encoding is shown in Figure 23. The Figure shows in particular that ACELP and TCX modes (Processors 23.015 and 23.016) are always in competition within a super-frame. Note however that the selector switch at the output of Processors 23.015 and 23.016 is such that each 20-ms frame within an 80-ms super-frame can be encoded in either ACELP mode or part of a TCX mode (either 20, 40 or 80 ms). The mode selection is as explained in the overview of the encoder.

The LF encoding therefore consists of two types of *modes*: an ACELP mode applied on 20-ms frames and TCX. To optimize the audio quality, the frame length of the TCX mode is allowed to be variable. The TCX mode operates on 20, 40 or 80-ms frames. The actual timing structure used in the encoder was shown in Figure 2.

In Figure 23, LPC analysis is first performed on the input LF signal noted $s(n)$. The window type, position and length for the LPC analysis is as shown in Figure 4, where the windows are positioned with respect to an 80-ms segment of LF signal, plus look-ahead. Note that the windows are positioned every 20 ms. After windowing, the LPC coefficients are computed (every 20 ms), then transformed into ISP representation and quantized for transmission to the decoder. The quantized ISP coefficients are interpolated every 5 ms to smooth the evolution of the spectral envelope. Processors 23.002 to 23.007 perform successively the windowing, autocorrelation, lag windowing and noise correction, Levinson-Durbin algorithm, ISP conversion, interpolation (in ISP domain) and computation of the interpolated LPC filters (in Processor 23.007, which outputs LPC parameters every 5 ms). Note that the ISP parameters are transformed again into ISF parameters (Processor 23.008) before quantization (Processor 23.009). The interpolated LPC parameters are noted $A(z)$, and the quantized version is noted $\hat{A}(z)$. The LF input signal ($s(n)$ in Figure 23) is then encoded both in ACELP mode (Processor 23.015) and in TCX mode (Processor 23.016), in all possible frame-length combinations as explained in the encoder overview and as shown in Figure 2. Note again that in ACELP mode, only 20-ms frames are considered in an 80-ms superframe, whereas in TCX mode, frames of 20, 40 and 80 ms are considered in Processor 23.016. Then, when all possible ACELP/TCX encoding combinations have been tried in Processors 23.015 and 23.016, all possible synthesis outputs (of Processors 23.015 and 23.016) are compared to the original signal in the weighted domain. It is important to note that in the final selection, there can be a mixture of ACELP and TCX frames in an encoded 80-ms super-frame, again as specified in the encoding possibilities shown in Figure 2. The error signals computed by Processor 23.019 are in the weighted domain: both the input LF signal (80-ms super-frame) and the synthesis output of Processors 23.015 and 23.016 are filtered with the perceptual filter formed by Processors 23.013 and 23.018 (identical processors, even if they have different ID numbers). For each possibility of the synthesis signal (again, possibly a mixture of ACELP and TCX frames), Processor 23.020 then computes the segmental Signal-to-Noise Ratio (SNR) over the whole 80-ms super-frame. The segmental SNR operated on 5-ms sub-frames. Computation of the segmental SNR is well known in the prior art. The mode combination which minimizes the segmental SNR over the entire 80-ms super-frame is then considered as the best encoding mode combination. Again, we refer to table 2 for all 26 possible mode combinations in a super-frame.

ACELP mode

The ACELP mode used in the illustrative embodiment is very similar to the ACELP algorithm operating at 12.8 kHz in the AMR-WB speech coding standard. The main changes compared to the ACELP algorithm in AMR-WB are:

- The LP analysis (a different windowing is used in the illustrative embodiment). The windowing used in the present invention for LPC analysis is shown in Figure 4.
- as well as the quantization of the codebook gains in every 5-ms sub-frame, as explained in the next section.

The ACELP mode operates on 5-ms sub-frames, where pitch analysis and algebraic codebook search are performed every sub-frame.

Codebook gain quantization in ACELP mode

In a given sub-frame of the ACELP mode, the two codebook gains (pitch gain g_p and code gain g_c) are quantized jointly based on the 7-bit gain quantization of AMR-WB. However, the Moving Average (MA) prediction of g_c, which is used in AMR-WB, is replaced in this invention by an absolute reference which is coded explicitly. Thus, the codebook gains are quantized here by a form of *mean*-removed quantization. This memoryless (non-predictive) quantization is well justified, because the ACELP mode may be applied to *non-speech* signals (e.g. transients in music), which requires a more general quantization than the predictive approach of AMR-WB which works well only for *speech* signals.

Computation and quantization of the absolute reference (in log domain):

A parameter, denoted μ_{ener}, is computed in open-loop and quantized once per frame with 2 bits. The current 20-ms frame of LPC residual $\mathbf{r} = (r_0, \ldots, r_L)$ is divided into 4 sub-frames, $\mathbf{r}_i=(r_i(0), \ldots, r_i(L_{sub}-1))$, with i=0..3. The parameter μ_{ener} is simply defined as the average of the sub-frame energies (in dB) over the current frame of the LPC residual:

$$\mu_{ener}(dB)=\frac{e_0(dB)+e_1(dB)+e_2(dB)+e_3(dB)}{4}$$

where

$$e_i=1+\frac{r_i(0)^2+\ldots+r_i(L_{sub}-1)^2}{L_{sub}}$$

is the energy of the i-th subframe of the LPC residual and $e_i(dB)=10\log_{10}\{e_i\}$. A constant 1 is added to the actual sub-frame energy in the above equation to avoid the subsequent computation of the logarithm of 0.

The mean μ_{ener} is then updated as follows:

$$\mu_{ener}(\text{dB}) := \mu_{ener}(\text{dB}) - 5 * (\rho_1 + \rho_2)$$

where ρ_i (I=1 or 2) is the normalized correlation computed as a side product of the *i*-th open-loop pitch analysis. This modification of μ_{ener} improves the audio quality for voiced speech segments.

The mean μ_{ener} (dB) is then scalar quantized with 2 bits. The quantization levels are set with a step of 12 dB to 18, 30, 42 and 54 dB. The quantization index can be simply computed as :

$$\text{tmp} = (\mu_{ener} - 18) / 12$$

$$\text{index} = \text{floor}(\text{tmp}+0.5)$$

$$\text{if (index} < 0) \text{ index} = 0, \text{ if (index} > 3) \text{ index} = 3$$

The reconstructed mean (in dB) is: $\hat{\mu}_{ener}(\text{dB}) = 18+(\text{index}*12)$.
However, the index and the reconstructed mean are then updated to improve the audio quality for transient signals such as attacks as follows:

$$max = \max(e_1(\text{dB}), e_2(\text{dB}), e_3(\text{dB}), e_4(\text{dB}))$$

$$\text{if } \hat{\mu}_{ener}(\text{dB}) < (max\text{-}27) \text{ and index} < 3,$$

$$\text{index} := \text{index} + 1,\ \hat{\mu}_{ener}(\text{dB}) := \hat{\mu}_{ener}(\text{dB}) + 1$$

Quantization of the codebook gains:

Recall that in AMR-WB, the gains (g_p and g_c) are quantized jointly in the form of (g_p, g_c * g_{c0}) where g_{c0} combines a MA prediction for g_c and a normalization with respect to the energy of the innovative code-vector.

In this invention, the two codebook gains (g_p and g_c) in a given sub-frame are jointly quantized with 7 bits exactly as in AMR-WB speech coding, in the form of (g_p, g_c*g_{c0}). The only difference lies in the computation of g_{c0}. The value of g_{c0} is based here on the quantized mean energy $\hat{\mu}_{ener}$ only, and computed as follows:

$g_{c0} = 10\text{^}((\hat{\mu}_{ener}(\text{dB}) - \text{ener}_c(\text{dB}))/20)$

where

$\text{ener}_c(\text{dB}) = 10 * \text{log10}(0.01 + (c(0)\text{^}2 + ... + c(Lsub-1)\text{^}2)/Lsub)$

TCX mode

In the TCX modes (Processor 23.016), an overlap with the next frame is defined to reduce blocking artifacts due to transform coding of the TCX target signal. The windowing and signal overlap depends both on the present frame type (ACELP or TCX) and size, and on the past frame type and size. The windowing used in the illustrative embodiment will be disclosed in the next section.

The TCX encoder employed in the illustrative embodiment is illustrated in Figure 6. We now disclose the TCX encoding procedure, and we will then go into more details about the lattice quantization used to quantize the spectrum.

TCX encoding in the illustrative embodiment proceeds as follows.

First, from Figure 6, the input signal is filtered through a weighting filter (Processors 6.001 and 6.002) to produce the weighted signal. Note that in TCX mode, the weighting filter uses the quantized LPC coefficients $\tilde{A}(z)$ instead of the unquantized $A(z)$ as in ACELP. This is because, contrary to ACELP which uses analysis-by-synthesis, the TCX decoder will have to perform the apply the inverse weighting filter to recover the excitation signal. If the previous encoded frame was ACELP, then the zero-input response (ZIR) of the weighting filter is removed from the weighted signal. In the illustrative embodiment, the ZIR is truncated to 10 ms and windowed in such a way that its amplitude monotonically decreases to zero at after 10 ms. Several time-domain windows can be used for this operation. The actual computation of this ZIR is not shown in Figure 6 since this signal, also referred to as the "filter ringing" in CELP-type coders, is well known to experts in the art. Once the weighted signal is computed, the signal is windowed in Processor 6.003, according to the window selection described in Figure 5.

After windowing by Processor 6.003, the windowed signal is transformed into the frequency-domain using an FFT (Processor 6.004).

Windowing in the TCX modes -- Processor 6.003

One of the key aspects of the invention is the mode switching between ACELP-type and TCX-type frames. To minimize the transition artifacts, proper care has to be given to windowing and overlap of successive frames. Adaptive windowing is performed by Processor 6.003. Figure 5 shows the window shapes depending on the TCX frame length and the type of the previous frame (ACELP of TCX). In Figure 5 (a), we first consider the case where the present frame is a TCX frame of length 20 ms. Depending on the past frame, the window applied can be :

1) **If the previous frame was an ACELP frame (of 20 ms duration)** : the window is a concatenation of two window segments – a flat window of duration 20 ms followed by the half-right portion of the square-root of a Hanning window of duration 2.5 ms – the encoder then needs a lookahead of 2.5 ms of the weighted speech

2) **If the previous frame was a TCX frame of 20 ms duration** : the window is a concatenation of three window segments – first, the left-half of the square-root of a Hanning window of 2.5 ms duration, then a flat window of duration 17.5 ms, then the half-right portion of the square-root of a Hanning window of duration 2.5 ms – the encoder again needs a lookahead of 2.5 ms of the weighted speech

3) **If the previous frame was a TCX frame of 40 ms duration** : the window is a concatenation of three window segments – first, the left-half of the square-root of a Hanning window of 5 ms duration, then a flat window of duration 15 ms, then the half-right portion of the square-root of a Hanning window of duration 2.5 ms – the encoder again needs a lookahead of 2.5 ms of the weighted speech

4) **If the previous frame was a TCX frame of 80 ms duration** : the window is a concatenation of three window segments – first, the left-half of the square-root of a Hanning window of 10 ms duration, then a flat window of duration 10 ms, then the half-right portion of the square-root of a Hanning window of duration 2.5 ms – the encoder again needs a lookahead of 2.5 ms of the weighted speech

In Figure 5 (b), we then consider the case where the present frame is a TCX frame of length 40 ms. Depending on the past frame, the window applied can be :

1) **If the previous frame was an ACELP frame (of 20 ms duration)** : the window is a concatenation of two window segments – a flat window of duration 40 ms followed by the half-right portion of the square-root of a Hanning window of duration 5 ms – the encoder then needs a lookahead of 5 ms of the weighted speech

2) **If the previous frame was a TCX frame of 20 ms duration** : the window is a concatenation of three window segments – first, the left-half of the square-root of a Hanning window of 2.5 ms duration, then a flat window of duration 37.5 ms, then the half-right portion of the square-root of a Hanning window of duration 5 ms – the encoder again needs a lookahead of 5 ms of the weighted speech

3) **If the previous frame was a TCX frame of 40 ms duration** : the window is a concatenation of three window segments – first, the left-half of the square-root of a Hanning window of 5 ms duration, then a flat window of duration 35 ms, then the half-right portion of the square-root of a Hanning window of duration 5 ms – the encoder again needs a lookahead of 5 ms of the weighted speech

4) **If the previous frame was a TCX frame of 80 ms duration** : the window is a concatenation of three window segments – first, the left-half of the square-root of the square-root of a Hanning window of 10 ms duration, then a flat window of duration 30 ms, then the half-right portion of the square-root of a Hanning window of duration 5 ms – the encoder again needs a lookahead of 5 ms of the weighted speech

Finally, in Figure 5 (c), we consider the case where the present frame is a TCX frame of length 80 ms. Depending on the past frame, the window applied can be :

1) **If the previous frame was an ACELP frame (of 20 ms duration)** : the window is a concatenation of two window segments – a flat window of duration 80 ms followed by the half-right portion of the square-root of a Hanning window of duration 5 ms – the encoder then needs a lookahead of 10 ms of the weighted speech

2) **If the previous frame was a TCX frame of 20 ms duration** : the window is a concatenation of three window segments – first, the left-half of the square-root of a Hanning window of 2.5 ms duration, then a flat window of duration 77.5 ms, then the half-right portion of the square-root of a Hanning window of duration 10 ms – the encoder again needs a lookahead of 10 ms of the weighted speech

3) **If the previous frame was a TCX frame of 40 ms duration** : the window is a concatenation of three window segments – first, the left-half of the square-root of a Hanning window of 5 ms duration, then a flat window of duration 75 ms, then the half-right portion of the square-root of a Hanning window of duration 10 ms – the encoder again needs a lookahead of 10 ms of the weighted speech

4) **If the previous frame was a TCX frame of 80 ms duration** : the window is a concatenation of three window segments – first, the left-half of the square-root of a Hanning window of 10 ms duration, then a flat window of duration 70 ms, then the half-right portion of the square-root of a Hanning window of duration 10 ms – the encoder again needs a lookahead of 10 ms of the weighted speech

We note again that all these window types are applied to the weighted signal, only when the present frame is a TCX frame. Frames of type ACELP are encoded as in the prior art describing AMR-WB encoding (i.e. through analysis-by-synthesis encoding of the excitation signal, so as to minimize the error in the target signal – the target signal is essentially the weighted signal to which the zero-input response of the weighting filter is removed). We note also that, when encoding a TCX frame that is preceded by another TCX frame, the windowed signal using the windows described above is quantized directly in a transform domain – as will be disclosed below. Then after quantization and inverse transformation, the synthesized weighted signal is recombined using overlap-and-add at the beginning of the frame with memorized look-ahead of the preceding frame.

On the other hand, when encoding a TCX frame preceded by an ACELP frame, the zero-input response of the weighting filter (actually, a windowed and truncated version of the zero-input response) is first removed from the windowed weighted signal : since the zero-input response is a good approximation of the first samples of the frame, the resulting effect is that the windowed signal will tend towards zero both at the beginning of the frame (because of the zero-input response subtraction) and at the end of the frame (because of the half-Hanning window applied to the look-ahead as described above and shown in Figure 5). Of course, the windowed and truncated zero-input response is added back to the quantized weighted signal after inverse transformation.

Hence, we achieve a suitable compromise between an optimal window (e.g. Hanning window) prior to the transform used in TCX frames, and the implicit rectangular window that has to be applied to the target signal when encoding in ACELP mode. This ensures a smooth switching between ACELP and TCX frames, while allowing proper windowing in both modes.

Time-frequency mapping – Processor 6.004

After windowing as described above, a transform is applied to the weighted signal in Processor 6.004. In the illustrative embodiment, a Fast Fourier Transform (FFT) is used.

Note (as shown in Figure 5) that TCX uses overlap between successive frames to reduce blocking artifacts. The length of the overlap depends on the length of the TCX modes: it is set respectively to 2.5, 5 and 10 ms when the TCX mode works with a frame length of 20, 40 and 80 ms (i.e. the length of the overlap is set to 1/8th of the frame length). This choice of overlap simplifies the radix in the fast computation of the DFT (by FFT). As a consequence the effective time support of the TCX modes is 22.5, 45 or 90 ms, as shown in Figure 2. With a sampling frequency of 12,800 samples per second (in the LF signal produced by Processor 1.001), and with frame+lookahead durations of 22.5, 45 or 90 ms, the time support of the FFT becomes 288, 576 or 1152 samples, respectively. These lengths can be expressed as 9 times 32, 9 times 64 and 9 times 128. Hence, a specialized radix-9 FFT can be used to computed rapidly the Fourier spectrum.

Pre-shaping (low-frequency emphasis) – Processor 6.005.

Once the Fourier spectrum (FFT) is computed, an adaptive low-frequency emphasis module is applied to the spectrum (Processor 6.005), to minimize the perceived distortion in the lower frequencies. The inverse low-frequency emphasis will be applied at the decoder, as well as in the encoder (Processor 6.007) to allow obtaining the excitation signal necessary to encode the next frames. The adaptive low-frequency emphasis is applied only on the first quarter of the spectrum, as follows.

First, we call X the transformed signal at the output of the FFT (Processor 6.004). The Fourier coefficient at Nyquist frequency is systematically set to 0. Then, if *N* is the number of samples in the FFT (*N* is thus the window length), the *K*=*N*/2 complex-valued Fourier coefficients are grouped in blocks of four consecutive coefficients, forming 8-dimensional real-valued blocks. Note that block lengths of size different than 8 can be used in general. In the illustrative embodiment, a block size of 8 is chosen to coincide with the 8-dimensional lattice quantizer used for spectral quantization. The energy of each block is computed, up to the first quarter of the spectrum. The energy *Emax* and position index *I* of the block with maximum energy are stored. Then, we calculate a factor *fm* for each 8-dimensional block with position index *m* smaller than *I*, as follows :

- calculate the energy *Em* of the 8-dimensional block at position index *m*
- compute the ratio *Rm* = *Emax* / *Em*
- compute the value $(Rm)^{1/4}$
- if *Rm* > 10, then set *Rm* = 10
- also, if *Rm* > *R*(*m*-1) then *Rm* = *R*(*m*-1)

This last condition ensures that the ratio function *Rm* decreases monotonically. Further, limiting the ratio *Rm* to be smaller or equal to 10 means that no spectral components in the low-frequency emphasis function will be modified by more than 20 dB.

After computing the ratio $Rm = (Emax / Em)^{1/4}$ for all blocks with position index smaller that *I* (and with the limiting conditions described above), we then apply these ratios as a gain for each corresponding block. This has the effect of increasing the energy of blocks with relatively low energy compared to the block with maximum energy *Emax*. Applying this procedure prior to quantization has the effect of shaping the coding noise in the lower band.

Figure 6a shows an example spectrum on which the above disclosed pre-shaping is applied. The frequency axis is normalized between 0 and 1, where 1 is the Nyquist frequency. The amplitude spectrum is shown in dB. In Figure 6a, the blue line is the amplitude spectrum before pre-shaping, and the red line portion is the modified (pre-shaped) spectrum. Hence, only the spectrum corresponding to the red line is modified in this example. In Figure 6a, the actual gain applied to each spectral component by the pre-shaping function is shown. We see that the gain is limited to 10, and monotonically decreases to 1 as it reaches the spectral component with highest energy (here, the third harmonic of the spectrum) at the normalized frequency of about 0.18.

Split multi-rate lattice vector quantization -- Processor 6.006

After low-frequency emphasis, the spectral coefficients are quantized using, in the illustrative embodiment, an algebraic quantizer based on lattice codes. The lattices used are 8-dimensional Gosset lattices, which explains the splitting of the spectral coefficients in 8-dimensional blocks. The quantization indices are essentially a global gain (from Processor 6.009) and a series of indices (from Processor 6.006) describing the actual lattice points used to quantize each 8-dimensional sub-vector in the spectrum. The lattice quantizer in Processor 6.006 performs (in a structured manner) a nearest neighbor search between each 8-dimensional vector of the scaled pre-shaped spectrum and the points in the lattice codebook used for quantization. The scale factor (global gain) actually determines the bit allocation and the average distortion. The larger the global gain, the more bits are used and the lower the average distortion. For each 8-dimensional vector of spectral coefficients, the lattice quantizer of Processor 6.006 outputs an index which indicates the lattice codebook number used and the actual lattice point chosen in the corresponding lattice codebook. The decoder will then be able to reconstruct the quantized spectrum using the global gain index along with the indices describing each 8-dimensional vector. The details of this procedure will be disclosed below.

Once the spectrum is quantized, the global gain (output of Processor 6.009) and lattice vectors indices (output of Processor 9.006) can be transmitted to the decoder.

Optimization of the global gain and computation of the noise-fill factor

A non-trivial step in using lattice vector quantizers is to determine the proper bit allocation within a pre-determined bit budget. Contrary to stored codebooks, where the index of a codebook point is basically its position in a table, the index of a lattice codebook point is calculated using mathematical (algebraic) formulae. The number of bits necessary to encode the lattice vector index is thus only known *after* the input vector is quantized. To insure staying within the pre-determined bit budget, this would in principle require trying several global gains and quantizing the normalized spectrum with each different gain to compute the total number of bits required. The global gain which achieves the bit allocation closest to the pre-determined bit budget, without exceeding it, would be chosen as the optimal gain. In this invention, a heuristic approach is used instead, to avoid having to quantize the spectrum several times before obtaining the optimum quantization and bit allocation.

For the sake of clarity, the key symbols related to this part of the illustrative embodiment of the invention are gathered in Table A-1.

Recall from Figure 6 that the time-domain TCX weighted signal **x** is processed by a transform **T** and a pre-shaping **P**, which produces a spectrum **X** to be quantized. In the illustrative embodiment of this invention, **T** is a FFT and the pre-shaping corresponds to the low-frequency enhancement disclosed above.

We will refer to vector **X** as the *pre-shaped spectrum*. We assume that this vector has the form $\mathbf{X} = [X_0\ X_1 \ldots X_{N-1}]^T$, where N is the number of transform coefficients obtained from **T** (the pre-shaping **P** does not change this number of coefficients).

Overview of the quantization procedure for the pre-shaped spectrum

In the illustrative embodiment of this invention, the pre-shaped spectrum **X** is quantized as described in Figure 7. The quantization is essentially based on the device of (Ragot, 2002). We assume an available bit budget of R_X bits for encoding **X**. As shown in Figure 7, **X** is quantized by *gain-shape split* vector quantization in three main steps:

- An estimated global gain g, called hereafter the *global gain*, is computed (Processors 7.001 and 7.002) and the spectrum **X** is normalized (Processor 7.003) by this factor to obtain $\mathbf{X'} = \mathbf{X}/g$. **X'** is thus the *normalized pre-shaped spectrum.*

- The multi-rate lattice vector quantization of (Ragot, 2002) (Processor 7.004) is applied to all 8-dimensional blocks of coefficients forming **X'**, and the resulting parameters are multiplexed. To be able to apply this quantization scheme, **X'** is divided into K sub-vectors of identical size, so that $\mathbf{X} = [\mathbf{X'}_0^T\ \mathbf{X'}_1^T \ldots \mathbf{X'}_{K-1}^T]^T$, where the kth sub-vector (or *split*) is given by

  $$\mathbf{X'}_k = [\, x'_{8k} \ldots x'_{8k+K-1} \,], \quad k = 0, 1, \ldots, K-1.$$

  Since the device of (Ragot, 2002) actually implements a form of 8-dimensional vector quantization, K is simply set to 8. We assume that N is a multiple of K.

- A noise fill-in gain *fac* is computed (in Processor 7.003) to later inject *comfort noise* in un-quantized splits of **X'**. The unquantized splits are blocks of coefficients which have been set to zero by the quantizer. The injection of noise allows to mask artifacts at low bit rates and improves audio quality. A single gain *fac* is used (as opposed to the prior art), because TCX coding assumes that the coding noise is flat in the target domain and shaped by the inverse perceptual filter $W(z)^{-1}$. Although pre-shaping is used here, the quantization and noise injection relies on the same principle.

As a consequence, the quantization of **X** shown in Figure 7 produces three kinds of parameters: the global gain *g*, the (split) algebraic VQ parameters and the noise fill-in gain *fac*. The bit allocation (or *bit budget*) R_X is decomposed as:

$$R_x = R_g + R + R_{fac},$$

where R_g, R and R_{fac} are the number of bits (or *bit budget*) allocated to *g*, algebraic VQ, and *fac*, respectively. In the illustrative embodiment, $R_{fac} = 0$.

Note that the multi-rate lattice vector quantization of (Ragot, 2002) is self-scalable and does not allow to control directly the bit allocation and the distortion in each split. This is the reason why the device of (Ragot, 2002) is applied to the splits of **X'** instead of **X**. The global gain *g* therefore plays a crucial role here, and its optimization controls the quality of the TCX mode. In the illustrative embodiment of this invention, the optimization of *g* is based on the log-energy of the splits.

In the sequel, each block of Figure 7 is detailed one by one.

Computing the Energy of Splits (Processor 7.001)

The energy (i.e. square-norm) of the split vectors plays a crucial role in the bit allocation algorithm, and is employed for determining the global gain (as well as the noise level). Recall that the *N*-dimensional input vector $\mathbf{X} = [x_0\ x_1 \ldots x_{N-1}]^T$ is partitioned into *K* splits, eight-dimensional subvectors, such that the *k*th split becomes $\mathbf{x}_k = [x_{8k}\ x_{8k+1} \ldots x_{8k+7}]^T$ for *k*

= 0, 1, ..., $\tilde{K}$ 1. It is assumed that N is a multiple of eight. The energy of the kth split vector is computed as

$$e_k = \mathbf{x}_k^T \mathbf{x}_k = x_{8k}^2 + \ldots + x_{8k+7}^2, \quad k = 0, 1, \ldots, \tilde{K}\ 1$$

Estimation of the global gain and noise level (Processor 7.002)

The global gain g controls directly the bit consumption of the splits and is solved from R(g) ≈ R, where R(g) is the number of bits used (or *bit consumption*) by all the split algebraic VQ for a given value of g. Recall that R is the bit budget allocated to the split algebraic VQ. As a consequence, the global gain g is optimized so as to match the *bit consumption* and the *bit budget* of algebraic VQ. The underlying principle is known as reverse water-filling in the literature.

To reduce the quantization complexity in the illustrative embodiment, the actual bit consumption for each split is not computed, but only estimated from the energy of the splits. This energy information together with an *a priori* knowledge of multi-rate RE_8 vector quantization allows to estimate R(g) as a simple function of g.

The global gain is determined by applying this basic principle in Processor 7.002. The bit consumption estimate of the split $\mathbf{X}_k$ is a function of the global gain g, and is denoted as $R_k(g)$. With the unity gain $g = 1$ we apply the heuristics

$$R_k(1) = 5 \log_2 (\varepsilon + e_k)/2, \quad k = 0, 1, \ldots, K-1$$

as a *bit consumption estimate*. The constant $\varepsilon > 0$ prevents the computation of $\log_2 0$, and the value $\varepsilon = 2$ is used. In general the constant ε is negligible compared to the energy of the split e_k.

The formula of $R_k(1)$ is based on *a priori* knowledge of the multi-rate quantizer of (Ragot, 2002) and the properties of the underlying RE_8 lattice:

- o For the codebook number $n_k > 1$, the bit budget requirement for coding the kth split at most $5n_k$ bits as can be confirmed from Table 1. This gives the factor 5 in the formula when $\log_2 (\varepsilon + e_k)/2$ is as an estimate of the codebook number.

- o The logarithm $\log_2$ reflects the property that the average square-norm of the codevectors is approximately doubled when using Q_{n_k} instead of Q_{n_k+1}. The property can be observed from Table 4.

- o The factor 1/2 applied to $\varepsilon + e_k$ calibrates the codebook number estimate for the codebook Q_2. The average square-norm of lattice points in this particular codebook is known to be around 8.0 (see Table 4). Since $\log_2 (\varepsilon + e_2))/2 \approx \log_2 (2 + 8.0))/2 \approx 2$, the codebook number estimation is indeed correct for Q_2.

Table 4
Some statistics on the square norms of the lattice points in different codebooks.

n	Average Norm
0	0
2	8.50
3	20.09
4	42.23
5	93.85
6	182.49
7	362.74

When a global gain g is applied to a split, the energy of $\mathbf{x}_k/g$ is obtained by dividing e_k by g^2. This implies the bit consumption of the gain-scaled split can be estimated based on $R_k(1)$ by subtracting $5 \log_2 g^2 = 10 \log_2 g$ from it:

$$R_k(g) = 5 \log_2 (\varepsilon + e_k)/2g^2$$

$$= 5 \log_2 (\varepsilon + e_k)/2 + 5 \log_2 g^2$$
$$= R_k(1) - g_{log} \qquad \text{(Eq. 4)}$$

in which $g_{log} = 10 \log_2 g$. The estimate $R_k(g)$ is lower bounded to zero, thus the relation

$$R_k(g) = \max\{R_k(1) - g_{log}, 0\} \qquad \text{(Eq. 5)}$$

is used in practice.

The bit consumption for coding all K splits is now simply a sum over the individual splits,

$$R(g) = R_0(g) + R_1(g) + \ldots + R_{K-1}(g). \qquad \text{(Eq. 6)}$$

The nonlinearity of equation 6 prevents solving analytically the global gain g that yields the bit consumption matching the given bit budget, $R(g) = R$. However, the solution can be found with a simple iterative algorithm because $R(g)$ is a monotonous function of g.

In this invention, the global gain g is searched efficiently by applying a bisection search to $g_{log} = 10 \log_2 g$, starting from the value $g_{log} = 128$. At each iteration iter, $R(g)$ is evaluated using (Eq. 4), (Eq. 5) and (Eq. 6), and g_{log} is respectively adjusted as $g_{log} := g_{log} \pm 128/2^{iter}$. Ten iterations give a sufficient accuracy. The global gain can then be solved from g_{log} as $g = 2^{g_{log}/10}$.

The flow chart of Figure 8 details the bisection algorithm employed for determining the global gain. The algorithm provides also the noise level as a side product. The algorithm starts by adjusting the bit budget R in Processor 8.001 to the value $0.95(R - K)$. The adjustment has been determined experimentally in order to avoid an over-estimation of the optimal global gain. The bisection algorithm requires as its initial value the bit consumption estimates $R_k(1)$ for $k = 0, 1, \ldots, K-1$ assuming a unity global gain. These estimates are computed employing equation 5 in Processor 8.003 having first obtained the square-norms of the splits e_k in

Processor 8.002. The algorithm starts from the initial values iter = 0, g_{log} = 0, and $\Delta = 128/2^{iter} = 128$ set in Processor 8.004.

Each iteration in the bisection algorithm comprises an increment $g_{log} := g_{log} + \Delta$ in Processor 8.006, and the evaluation of the bit consumption estimate $R(g)$ in Processors 8.007 and 8.008 with the new value of g_{log}. If the estimate $R(g)$ exceeds the bit budget R in Processor 8.009, the update of g_{log} is reversed in Processor 8.011. The iteration ends by incrementing the counter iter and halving the step size Δ in Processor 8.010. After ten iterations, a sufficient accuracy for g_{log} is obtained and the global gain can be solved $g = 2^{g_{log}/10}$ in Processor 8.012. The noise level g_{ns} is estimated in Processor 8.013 by averaging the bit consumption estimates of those splits that are likely to be left unquantized with the determined global gain g_{log}.

Figure 9 details the steps involved in determining the noise level *fac*. The noise level is computed as the square root of the average energy of the splits that are likely to be left un-quantized. For a given global gain g_{log}, a split is said to be likely to be un-quantized if its estimated bit consumption is less than 5 bits, i.e. if $R_k(1) - g_{log} < 5$. The total bit consumption of all such splits, $R_{ns}(g)$, is obtained by summing $R_k(1) - g_{log}$ over the splits for which $R_k(1) - g_{log} < 5$. The average energy of these splits can then be computed in log domain from $R_{ns}(g)$ as $R_{ns}(g)/nb$, where *nb* is the number of these splits. The noise level is

$$fac=2^{Rns(g)/nb-5}$$

In this equation, the constant -5 in exponent is a (conservative) tuning factor which adjusts the noise factor 3 dB (in energy) below the real estimated based on the average energy.

Multi-Rate Lattice Vector Quantization (Processor 6.006)

The basic building block or Processor 6.006 is the multi-rate quantization means disclosed and detailed in (Ragot, 2002) is applied. The eight-dimensional splits of the normalized spectrum **X'** are coded with multi-rate quantizer that employs a set of RE_8 codebooks denoted as $\{Q_0, Q_2, Q_3, \ldots\}$. In the illustrative embodiment of the invention, the codebook Q_1 is not defined in the set in order to improve coding efficiency. The nth codebook is denoted by Q_n where n is referred to as a *codebook number*. All codebooks Q_n are constructed as subsets of the same 8-dimensional RE_8 lattice, $Q_n \subset RE_8$. The bit rate of the nth codebook defined as bits per dimension is 4n/8, i.e. each codebook Q_n contains 2^{4n} code-vectors. The construction of the multi-rate quantizer follows the before-mentioned reference.

For the kth eight-dimensional split $\mathbf{X'}_k$, the encoder of the multi-rate quantizer finds the nearest neighbor $\mathbf{Y}_k$ in RE_8, and outputs

- the smallest codebook number n_k such that $Y_k \in Q_{n_k}$, and
- the index i_k of $\mathbf{Y}_k$ in Q_{n_k}.

The codebook number n_k is a side information that has to be made available to the decoder together with the index i_k to reconstruct the codevector $\mathbf{Y}_k$. By construction of the multi-rate quantizer, the size of index i_k is $4n_k$ bits for $n_k > 1$. This index can be represented with 4-bit blocks.

For $n_k = 0$, the reconstruction $\mathbf{y}_k$ becomes an eight-dimensional zero vector and i_k is not needed.

Handling of Bit Budget Overflow and Indexing of Splits (Processor 7.005)

For a given global gain g, the real bit consumption may either exceed or remain under the bit budget. In this invention, a possible bit budget underflow is not addressed by any specific means, but the available extra bits are zeroed and left unused. When a bit budget overflow occurs, the bit consumption is accommodated into the bit budget R_x in Processor 7.005 by zeroing some of the codebook numbers n_0, n_1, ..., n_{K-1}. Zeroing a codebook number $n_k > 0$ reduces the total bit consumption at least by $5n_k$-1 bits. The splits zeroed in the handling of the bit budget overflow are reconstructed at the decoder by noise fill-in.

To minimize the coding distortion that occurs when the codebook numbers of some splits are forced to zero, these splits shall be selected prudently. In a illustrative embodiment of the invention, the bit consumption is accumulated by handling the splits one by one in an descending order of their energy $e_k = \mathbf{x}_k^T\mathbf{x}_k$ for $k = 0, 1, ..., K-1$. This procedure is signal dependent and in agreement with the means used earlier in determining the global gain.

Before examining the details of overflow handling in module 7.005, it is advisable to recall the structure of the code used for representing the output of the multi-rate quantizers. The unary code of $n_k > 0$ comprises $k-1$ ones followed by a zero stop bit. As was shown in Table 1, $5n_k - 1$ bits are needed to code the index i_k and the codebook number n_k excluding the stop bit. The codebook number $n_k = 0$ comprises only a stop bit indicating zero split. When K splits are coded, at maximum of only $K-1$ stop bits are needed as the last one is implicitly determined by the bit budget R and thus redundant. More specifically, when k last splits are zero, only $k-1$ stop bits suffice because the last zero splits can be decoded by knowing R.

The overflow handling module 7.005 implemented in accordance with a illustrative embodiment is depicted in the functional block diagram of Figure 10. The procedure operates with split indices $\kappa(0)$, $\kappa(1)$, ..., $\kappa(K-1)$ determined in Processor 10.001 by sorting the square-norms of splits in a descending order such that $e_{\kappa(0)} \geq e_{\kappa(1)} \geq ... \geq e_{\kappa(K-1)}$. Thus the index $\kappa(0)$ refers to the split $\mathbf{x}_{\kappa(k)}$ that has the kth largest square-norm. The square norms of splits are supplied to overflow handling as an output of module 7.001.

The kth iteration of overflow handling can be skipped readily if $n_{\kappa(k)} = 0$ by continuing directly to the next iteration because zero splits cannot cause an overflow. This functionality is implemented with logic block 10.005. Assuming now that the $\kappa(k)$th split is non-zero, the RE_8 point $\mathbf{y}_{\kappa(k)}$ is first indexed in block 7.004. The multirate indexing provides the exact value of the codebook number $n_{\kappa(k)}$ and code-vector index $i_{\kappa(k)}$. The bit consumption of all splits up to and including the current $\kappa(k)$th split can be calculated.

Using the properties of the unary code, the bit consumption R_k up to and including the current split is counted in block 10.008 as a sum of two terms: the $R_{D,\,k}$ bits needed for the data excluding stop bits and the $R_{S,\,k}$ stop bits:

$$R_k = R_{D,\,k} + R_{S,\,k}, \qquad \text{(Eq. 7)}$$

where for $n_{\kappa(k)} > 0$

$$R_{D,\,k} = R_{D,\,k-1} + 5n_{\kappa(k)} - 1, \qquad \text{(Eq. 8)}$$

$$R_{S,\,k} = \max\{\kappa(k), R_{S,\,k-1}\}. \qquad \text{(Eq. 9)}$$

The required initial values are set to zero. The stop bits are counted in Processor 10.007 from Equation (9) taking into account that only splits up to the last non-zero split so far must be indicated with stop bits, because the subsequent splits are known to be zero by construction of the code. The index of the last non-zero split can also be expressed as $\max\{\kappa(0), \kappa(k), \ldots, \kappa(k)\}$.

Since the overflow handling starts from zero initial values for $R_{D,\,k}$ and $R_{S,\,k}$ in (Eq. 8) and (Eq. 9), the bit consumption up to the current split fits always into the bit budget, $R_{S,\,k-1} + R_{D,\,k-1} < R$. If the bit consumption R_k including the current $\kappa(k)$th split exceeds the bit budget R as verified in logic block 10.008, the codebook number $n_{\kappa(k)}$ and reconstruction $\mathbf{y}_{\kappa(k)}$ are zeroed in block 10.009. The bit consumption counters $R_{D,\,k}$ and $R_{D,\,k}$ are accordingly reset to their previous values in block 10.010. After this, the overflow handling can proceed to the next iteration by incrementing *k* and continuing from logic block 10.003.

Note that block 10.004 produces the indexing of splits as an integral part of the overflow handling routines. The indexing can be stored and supplied further to the bit stream multiplexer module.

Quantized spectrum de-shaping – Processor 6.007

Once the spectrum is quantized using the split multi-rate lattice VQ of Processor 6.006, the quantization indices (codebook numbers and lattice point indices) can be calculated and sent to the channel. Nearest neighbor search in the lattice, and index computation, are performed as in (Ragot, 2002). The TCX encoder then performs spectrum de-shaping in Processor 6.007, in such a way as to invert the pre-shaping of Processor 6.005.

Spectrum de-shaping operates using only the quantized spectrum. To obtain a process that inverts the steps of Processor 6.005, Processor 6.007 applies the following steps :

- calculate the position *I* and energy *Emax* of the 8-dimensional block of highest energy in the first quarter (low frequencies) of the spectrum
- calculate the energy *Em* of the 8-dimensional block at position index *m*
- compute the ratio $Rm = Emax / Em$
- compute the value $(Rm)^{1/2}$

- if *Rm* > 10, then set *Rm* = 10
- also, if *Rm* > *R*(*m*-1) then *Rm* = *R*(*m*-1)

After computing the ratio *Rm* = *Emax* / *Em* for all blocks with position index smaller that *I*, we then apply the multiplicative inverse of this ratio as a gain for each corresponding block. Note the major differences with the pre-shaping of Processor 6.005 : 1) in the de-shaping of Processor 6.007, we compute the square-root (and not the power ¼) of the ratio *Rm* and 2) this ratio is taken as a divider (and not a multiplier) of the corresponding 8-dimensional block. If the effect of quantizer in Processor 6.006 is neglected (perfect quantization), it can be shown that the output of Processor 6.007 is exactly equal to the input of Processor 6.005. The pre-shaping process is thus an invertible process.

HF encoding

The encoding of the HF signal of Processor 1.003 is detailed in Figure 11. Recall from Figure 1 that the HF signal is composed of the frequency components above 6400 Hz in the input signal. The bandwidth of this HF signal depends on the input signal sampling rate. To encode the HF signal at a low rate, a bandwidth extension (BWE) approach is employed in the illustrative embodiment. In BWE, energy information is sent to the decoder in the form of spectral envelope and frame energy, but the fine structure of the signal is extrapolated at the decoder from the received (decoded) excitation signal in the Lf signal, which, in the present invention, was encoded in the switched ACELP/TCX encoder in Processor 1.002.

The down-sampled HF signal (output of Processor 1.001) is called $s_{HF}(n)$ In Figure 11. The spectrum of this signal can be seen as a folded version of the high-frequency band prior to down-sampling. An LPC analysis is performed on $s_{HF}(n)$ to obtain a set of coefficients which model the spectral envelope of this signal. Typically, fewer parameters are necessary than in the LF signal. In this invention, we use a filter of order 8. The LPC coefficients are then transformed into ISP representation and quantized for transmission. The number of LPC analysis in an 80-ms super-frame depends on the frame lengths in the super-frame. The ISP coefficients are then interpolated in Processor 11.005.

We recall that a set of LPC filter coefficients can be represented as a polynomial in the variable z. Then, we call $A(z)$ the LPC filter for the LF signal and $A_{HF}(z)$ the LPC filter for the HF signal. Their quantized versions are respectively $\hat{A}(z)$ and $\hat{A}_{HF}(z)$. From the LF signal ($s(n)$ in Figure 11), a residual signal is first obtained by filtering $s(n)$ through the residual filter $\hat{A}(z)$ in Processor 1.014. Then, this residual is filtered through the quantized HF synthesis filter, $1/\hat{A}_{HF}(z)$. Up to a gain factor, this produces a good approximation of the HF signal, but in a spectrally folded version. The actual HF synthesis signal will be recovered when up-sampling is applied to this signal

Since the excitation is taken from the LF signal, an important step is to compute the proper gain for the HF signal. This is done by comparing the energy of the reference HF signal ($s_{HF}(n)$) with the energy of the synthesized HF signal. The energy is computed once per 5-ms subframe, with energy match ensured at the 6400 Hz subband boundary. Specifically, the synthesized HF signal and the reference HF signal are filtered through a perceptual filter. In the illustrative embodiment, this perceptual filter is derived from $A_{HF}(z)$ and is called "HF perceptual filter" in Figure 11. The ration of the energy of these two filtered signals is computed every 5 ms, and expressed in dB. There are 4 such gains in a 20-ms frame (one for every 5-ms subframe). This 4-gain vector represents the gain that should be applied to the HF signal to properly match the HF signal energy. Instead of transmitting this gain directly, an estimated gain ratio is first computed by comparing the gains of filters $\hat{A}(z)$ from the lower band and $\hat{A}_{HF}(z)$ from the higher band. This gain ratio estimation is detailed in Figure 11a and will be explained below. The gain ratio estimation is interpolated every 5-ms, expressed in dB and subtracted in Processor 11.010 from the measured gain ratio. The resulting gain differences or gain corrections, noted $\bar{g}_0$ to $\bar{g}_{nb-1}$ in Figure 11, are quantized in Processor 11.009. In the illustrative embodiment, the gain corrections are quantized as 4-dimensional vectors, i.e. 4 values per 20-ms frame.

The gain estimation computed in Processor 11.007 from filters $\hat{A}(z)$ and $\hat{A}_{HF}(z)$ is detailed in Figure 11a. These two filters are available ate the decoder side. The first 64 samples of a decaying sinusoid at Nyquist frequency π radians per sample is first computed by filtering a unit impulse through a one-pole filter (Processor J01). The Nyquist frequency is used since the goal is to match the filter gains at around 6400 Hz, i.e. at the junction frequency between the LF and HF signals. Note the the 64-sample length of this reference signal is the sub-frame length (5 ms). The decaying sinusoid is then filtered first through $\hat{A}(z)$, in Processor J02, to obtain a low-frequency residual, then through $1/\hat{A}_{HF}(z)$ in Processor J03 to obtain a synthesis signal from the HF synthesis filter. We note that if filters $\hat{A}(z)$ and $\hat{A}_{HF}(z)$ have identical gains at the normalized frequency of π radians per sample, the energy of the output of Processor J03 would be equivalent to the energy of the input of Processor J02 (the decaying sinusoid). If the gains differ, then this gain difference is taken into account in the energy of the signal at the output of Processor J03, noted $x(n)$. The correction gain should actually increase as the energy of $x(n)$ decreases. Hence, the gain correction is computed in Processor J04 as the multiplicative inverse of the energy of signal $x(n)$, in the logarithmic domain (i.e. in dB). To get a true energy ratio, the energy of the decaying sinusoid (output of Processor J01), in dB, should be removed from the output of Processor J04. However, since this energy offset is a constant, it will simply be taken into account in the gain correction encoder in Processor 11.009.

At the decoder, the gain of the HF signal can be recovered by adding the output of Processor 1.008 (known at the decoder) to the decoded gain corrections (encoded in Processor 11.009).

DETAILED DESCRIPTION OF THE DECODER

The role of the decoder is to read the encoded parameters from the bitstream and synthesize a reconstructed audio super-frame. A high-level block diagram of the decoder is shown in Figure 12.

Recall that each 80-ms super-frame is encoded into four successive binary packets of equal size. These four packets form the input of the decoder. Since all packets may not be available due to channel erasures, the main demultiplexer (Processor 12.001) also gets as an input four bad frame indicators **BFI** = (bfi_0, bfi_1, bfi_2, bfi_3) which tell which of the four packets have been received. We assume here that $bfi_k = 0$ when the k-th packet is received, and $bfi_k = 1$ when the k-th packet is lost. The size of the 4 packets is specified to Processor 12.001 by the input *bit_rate_flag* (which indicates the bit rate used by the encoder).

Main demultiplexing

The demultiplexer (Processor 12.001) simply does the reverse operation of the multiplexer. The bits related to the encoded parameters in packet k are extracted when packet k is available (i.e. $bfi_k = 0$).

Recall that the encoded parameters are divided into 3 categories: mode indicators, low-frequency (LF) parameters and high-frequency (HF) parameters. The mode indicators specify which encoding mode was used at the encoder (ACELP or TCX-20/40/80). After the main demultiplexer (Processor 12.001), these parameters are decoded by mode extrapolation (Processor 12.002), ACELP/TCX decoding (Processor 12.003) and HF decoding (Processor 12.004), successively. This decoding results into 2 signals, a LF synthesis and a HF synthesis, which are combined to form the audio output in Processor 12.005. We assume that an input flag **FS** indicates to the decoder what is the output sampling rate (in the illustrative embodiment the allowed sampling rates are 16 kHz and above).

The processors in Figure 12 following the main demultiplexer are presented in details in the next paragraphs.

Mode extrapolation

In the presence of packet losses, the decoder tries to recover the missing mode indicators from the available ones (including also mode indicators of previous superframes). Recall that the mode selected in a given super-frame is given by **MODE** = (m_0, m_1, m_2, m_3) where $0 \leq m_k \leq 3$ and k=0,..,3. The 26 valid modes are enumerated in Table 2. When $bfi_k = 1$, the value m_k is not available and has to be estimated from other received information.

In the illustrative embodiment, the mode extrapolation is essentially a mode repetition. The mode indicators from the previous super-frame only are reused in the extrapolation. More precisely, only the last indicator of the previous mode is used. Hence, the mode of the previous superframe is seen as (x, x, x, m_{-1}) where the value x is not relevant (this value is not used here) and $0 \leq m_{-1} \leq 3$ is the final indicator of the previous mode. Note that if m_{-1} was not available, the extrapolated value of m_{-1} is used.

A high-level description of the mode extrapolation device is given in Figure 13. Based on the values in **BFI**, the available mode indicators are set from the bits coming from the demultiplexer (Processor 13.001) and the number of packet losses n_{loss} is counted in Processor 13.002. In Processor 13.001 the mode is given by **MODE** = (m_0, m_1, m_2, m_3) with $0 \leq m_k \leq 3$ when the indicator m_k is available (i.e. $bfi_k = 0$), and $m_k = -1$ when $bfi_k = 1$. Then, the missing mode indicators (for which $m_k = -1$) are extrapolated in Processor 13.003. The logic of Processor 13.003 is shown in the flow chart of Figure 14. Since the latter figure is quite self-explanatory, we focus here on the rationale behind the related processing:

- There exists redundancy in the definition of mode indicators. A TCX-80 frame is described by **MODE** = (3,3,3,3), and a TCX-40 frame is described by (2,2,x,x) or (x,x,2,2). Therefore, *in the absence of bit errors*, the mode indicators describing a TCX-40 or TCX-80 frame can be easily extrapolated in case of partial packet losses, when a single value m_k = 2 or 3 is available (this is done in Processors 14.005, 14.007 and 14.009).

- o The frame-erasure concealment in ACELP relies on the pitch delay and codebook gains of the previous ACELP frame. However in switched ACELP/TCX coding there is no guarantee that the frame preceding an ACELP frame was also encoded by ACELP. Assuming that m_k is not available and that the extrapolation has to choose between $m_k = 0$ or $m_k = 1$, the extrapolation will select ACELP decoding ($m_k = 0$) only if $m_{k-1} = 0$ (Processor 14.013). Otherwise the ACELP parameters needed for concealment would not be up-to-date. As a consequence, under the above assumptions, if $m_{k-1}>0$, the value $m_k = 1$ will be selected (Processor 14.014).

- o If 3 packets are lost and if the only available mode indicator is m_k = 3 with k =0,1,2 or 3, a mode (3,3,3,3) corresponding to TCX-80 should normally be extrapolated. Yet, with the bitstream format used in the illustrative embodiment, losing 3 packets out of 4 in TCX-80 means

    1) losing roughly 3 quarters of the TCX target spectrum and

    2) having no information about the TCX global gain since the gain repetition in TCX-80 is designed to perform well for up to 2 packet losses.

  As a consequence, the mode (3,3,3,3) is rather replaced by the mode (1,1,1,1) in the extrapolation when more than 2 packets are lost (Processor 14.004). Note that this causes the concealment of

TCX-20 to be used (the synthesis will actually be progressively faded out).

There could be alternative procedures for the mode extrapolation procedure illustrated in Figure 14. However the procedure disclosed above has the advantage of being simple and to minimize decoding complexity by avoiding additional signal analysis. The handling of bit errors on mode indicators is also implicit, although suboptimal.

Decoding of the low-frequency (LF) signal: ACELP/TCX decoding

After extrapolating the missing mode indicators, the extrapolated **MODE** is used to demultiplex and decode the rest of the bitstream (based on **BFI**).

The decoding of the LF signal involves essentially ACELP/TCX decoding. This procedure is described in more details in Figure 15. The ACELP/TCX demultiplexer (Processor 15.001) extracts the (encoded) LF parameters based on the values of **MODE** . These parameters are split into ISF parameters on the one hand and ACELP- or TCX-specific parameters on the other hand.

The decoding of the LF parameters is controlled by Processor 15.002. In particular, this processor sends control signals to the ISF decoding (Processors 15.003), the ISP interpolation (Processor 15.004), as well as ACELP and TCX decoders (Processors 15.007, 15.008). It also handles the switching between the ACELP decoder (Processor 15.007) and the TCX decoder (Processor 15.008) by setting proper inputs to these two decoders and activating the switch selector at the output (Processor 15.009). It also controls the output buffer of the LF signal (Processor 15.010) so that the ACELP or TCX decoded frames are written in the right time segments of the 80-ms output buffer.

Processor 15.002 generates control data which are internal to the LF decoder : **BFI_ISF**, *nb* (the number of subframes for ISP interpolation), *bfi_acelp*, L_{TCX} (TCX frame length), **BFI_TCX**, *switch_flag*, and *frame_selector* (to set a frame pointer on the output buffer). The nature of these data is defined in more details below:

➤ **BFI_ISF** can be expanded as the 2-D integer vector **BFI_ISF** = (bfi_{1st_stage} bfi_{2nd_stage}) and consists of bad frame indicators for ISF decoding. The value bfi_{1st_stage} is binary, and $bfi_{1st\_stage} = 0$ when the ISF 1st stage is available and $bfi_{1st\_stage} = 1$ when it is lost. The value $0 \leq bfi_{2nd\_stage} \leq 31$ is a 5-bit flag providing a bad frame indicator for each of the 5 splits of the ISF 2nd stage : $bfi_{2nd\_stage} = bfi_{1st\_split} + 2 * bfi_{2nd\_split} + 4 * bfi_{3rd\_split} + 8 * bfi_{4th\_split} + 16 * bfi_{5th\_split}$, where $bfi_{kth\_split} = 0$ when split k is available, 1 otherwise. With the bitstream format used in the illustrative embodiment, the values of bfi_{1st_stage} and bfi_{2nd_stage} can be computed from **BFI** = (bfi_0 bfi_1 bfi_2 bfi_3) as follows :

For ACELP or TCX-20 in packet *k*, **BFI_ISF** = (bfi_k),

For TCX-40 in packets *k* and *k+1*, **BFI_ISF** = (bfi_k $(31 * bfi_{k+1})$),

For TCX-80 (in packets k=*0 to 3)*, **BFI_ISF** = (bfi_0 $(bfi_1+6*bfi_2+20*bfi_3)$)

These values of **BFI_ISF** can be explained directly by the bitstream format used to pack the bits of ISF quantization, and how the stages and splits are distributed in one or several packets depending on the coder type (ACELP/TCX-20, TCX-40 or TCX-80).

➤ The number of subframes for ISF interpolation refers to the number of 5-ms subframes in the ACELP or TCX decoded frame. Thus, *nb* = 4 for ACELP and TCX-40, 8 for TCX-40 and 16 for TCX-80.

- *bfi_acelp* is a binary flag indicating an ACELP packet loss. It is simply set as *bfi_acelp* = bfi_k for an ACELP frame in packet *k*.

- The TCX frame length (in samples) is given by L_{TCX} = 256 (20 ms) for TCX-20, 512 (40 ms) for TCX-40 and 1024 (80 ms) for TCX-80. Note that this does not take into account the overlap used in TCX to reduce blocking effects.

- **BFI_TCX** is a binary vector used to signal packet losses to the TCX decoder : **BFI_TCX** = (bfi_k) for TCX-20 in packet *k*, (bfi_k bfi_{k+1}) for TCX-40 in packets *k* and *k+1*, and **BFI_TCX** = **BFI** for TCX-80.

The other data generated by Processor 15.002 are quite self-explanatory. The switch selector activates Processor 15.009 according to the type of decoded frame (ACELP or TCX). The frame selector allows to write the decoded frames (ACELP or TCX-20, TCX-40 or TCX-80) into the right 20-ms segments of the superframe. Note that in Figure 15 some auxiliary data also appear (**ACELP_ZIR**, rms_{wsyn}). These data are defined in subsequent paragraphs and they are not essential for understanding the device illustrated in Figure 15.

Processor 15.003 corresponds to the ISF decoder defined in the AMR-WB speech coding standard (with the same MA prediction and quantization tables) except for the handling of bad frames. The only difference compared to the reference AMR-WB device is the use of **BFI_ISF** = (bfi_{1st_stage} bfi_{2nd_stage}) instead of a single binary bad frame indicator. When the 1st stage of the ISF quantizer is lost (i.e. , bfi_{1st_stage} =1) the ISF parameters are simply decoded using the frame-erasure concealment of the AMR-WB ISF decoder. When the 1st stage is available (i.e. , bfi_{1st_stage} =0), this 1st stage is decoded. The 2nd stage split vectors are accumulated to the decoded 1st stage only if they are available. The reconstructed ISF residual is added to the MA prediction and the ISF mean vector to form the reconstructed ISF parameters.

Processors 15.004 is a conversion of ISF parameters (defined in the frequency domain) into ISP parameters (in the cosine domain). This operation is taken from AMR-WB speech coding.

Processor 15.005 realizes a simple linear interpolation between the ISP parameters of the previous decoded frame (ACELP/TCX-20, TCX-40 or TCX-80) and the decoded ISP parameters. The interpolation is conducted in the ISP domain and results in ISP parameters for each 5-ms subframe, according to the formula:

$$\mathbf{isp}_{subframe\text{-}i} = i/nb \; * \; \mathbf{isp}_{new} + (1\text{-}i/nb) * \mathbf{isp}_{old},$$

where *nb* is the number of subframes in the current decoded frame (*nb*=4 for ACELP and TCX-20, 8 for TCX-40, 16 for TCX-80), *i*=0,...,*nb*-1 is the subframe index, $\mathbf{Isp}_{old}$ is the set of ISP parameters obtained from the decoded ISF parameters of the previous decoded frame (ACELP, TCX-20/40/80) and $\mathbf{isp}_{new}$ is the set of ISP parameters obtained from the ISF parameters decoded in Processors 15.003. The interpolated ISP parameters are then converted into linear-predictive coefficients for each subframe in Processor 15.006.

The ACELP and TCX decoders (Processors 15.007 and 15.008) will be detailed separately at the end of the overall ACELP/TCX decoding description.

ACELP/TCX switching

The description of Figure 15 in the form of a block diagram is completed by the flow chart of Figure 16, which defines exactly how the switching between ACELP and TCX is handled based on the super-frame mode indicators in **MODE**. Therefore Figure 16 explains how the Processors 15.003 to 15.006 of Figure 15 are used.

One of the key aspects of ACELP/TCX decoding is the handling of an *overlap* from the past decoded frame to enable seamless switching between ACELP and TCX as well as between TCX frames. Figure 16 presents this key feature in details (for the decoding side only).

The *overlap* consists of a single 10-ms buffer: **OVLP_TCX**. When the past decoded frame is ACELP, **OVLP_TCX = ACELP_ZIR** memorizes the zero-impulse response (ZIR) of the LP synthesis filter ($1/A(z)$) in weighted domain of the previous ACELP frame. When the past decoded frame is TCX, only the first 2.5 ms (32 samples) for TCX-20, 5 ms (64 samples) for TCX-40, 10 ms (128 samples) for TCX-80 of are used in **OVLP_TCX** (the other samples are set to zero).

As illustrated in Figure 16, the ACELP/TCX decoding relies on a sequential interpretation of the modes indicators in **MODE**. The packet number and decoded frame index k is incremented from 0 to 3. The loop realized by Processors 16.002, 16.003 and 16.021 to 16.023 allows to sequentially process the 4 packets of a 80-ms superframe. The description of Processors 16.005, 16.006 and 16.009 to 16.011 is skipped because they realize the ISF decoding, ISF to ISP conversion, ISP interpolation and ISP to $A(z)$ conversion described previously.

When decoding ACELP (i.e. when m_k=0), the buffer **ACELP_ZIR** is updated and the length *ovp_len* of the TCX overlap is set to 0 (Processors 16.013 and 16.017). The actual calculation of **ACELP_ZIR** is detailed in the next paragraph dealing with ACELP decoding.

When decoding TCX, the buffer **OVLP_TCX** is updated (Processors 16.014 to 16.016) and the actual length *ovp_len* of the TCX overlap is set to a number of samples equivalent to 2.5, 5 and 10 ms for TCX-20, 40 and 80, respectively (Processors 16.018 to 16.020). The actual calculation of **OVLP_TCX** is detailed in the next paragraph dealing with TCX decoding.

Note that the ACELP/TCX decoder also computes two parameters for subsequent pitch post-filtering of the LF synthesis. the pitch gains $\mathbf{g}_p = (g_0, g_1, \ldots, g_{15})$ and pitch lags $\mathbf{T} = (T_0, T_1, \ldots, T_{15})$ for each 5-ms subframe of the 80-ms superframe. These parameters are initialized in Processor 16.001. For each new superframe, the pitch gains are set by default to $g_{pk} = 0$ for k=0,...,15, while in the pitch lags are all initialized to 64 (i.e. 5 ms). These vectors are modified only by ACELP (in Processor 16.013) : if ACELP is defined packet k, $g_{4k}, g_{4k+1}, \ldots, g_{4k+3}$ correspond to the pitch gains in each decoded ACELP subframe, while $T_{4k}, T_{4k+1}, \ldots, T_{4k+3}$ are the pitch lags.

ACELP decoding

The ACELP decoder presented in Figure 17 is derived from the AMR-WB speech coding algorithm (Bessette et al, 2002). The new or modified blocks compared to the ACELP decoder of AMR-WB are highlighted (by shading these blocks) in Figure 17.

As illustrated in Figure 17, ACELP decoding consists of reconstructing the excitation signal $r(n)$ in Processor 17.015 as the linear combination $g_p\, p(n) + g_c\, c(n)$, where g_p and g_c are respectively the pitch gain and the fixed-codebook gain, T the pitch lag, and $p(n)$ and $c(n)$ are respectively pitch contribution derived from the *adaptive codebook* (Processor 17.005) and a post-processed codevector of the *innovative codebook* (Processors 17.009). Note that when the pitch lag T is fractional, $p(n)$ involves interpolation in Processor 17.005. Then, the reconstructed excitation is passed through the *synthesis filter* $1/\hat{A}(z)$ (Processor 17.016) to obtain the synthesis. This processing is done per sub-frame based on the interpolated LP coefficients and the synthesis is buffered in Processor 17.017. The whole ACELP decoding process is controlled by Processor

17.002. Packet erasures (signalled by *bfi_acelp* = 1) are handled by switching from the innovative codebook to a random innovative codebook (Processors 17.010), extrapolating pitch and gain parameters from their past values in Processors 17.003 and 17.004, and relies on the extrapolated LP coefficients.

The significant changes compared to the ACELP decoder of AMR-WB are restricted to the gain decoding (Processor 17.003), the computation of the zero-impulse response (ZIR) of $1/\hat{A}(z)$ in weighted domain (Processors 17.018 to 17.020) and the update of the r.m.s value of the weighted synthesis (rms_{wsyn}) in Processors 17.021 and 17.022. The gain decoding has been already disclosed when *bfi_acelp* = 0 or 1. It is based on a mean energy parameter so as to apply mean-removed VQ.

The ZIR of $1/\hat{A}(z)$ is computed here in weighted domain for switching from an ACELP frame to a TCX frame while avoiding blocking effects. The related processing is broken into 3 steps and its result is stored in a 10-ms buffer denoted by **ACELP_ZIR** :

1) the computation of the 10-ms ZIR of $1/\hat{A}(z)$ where the LP coefficients are taken from the last ACELP subframe (Processor 17.018),

2) perceptual weighting of the ZIR (Processor 17.019),

3) **ACELP_ZIR** is found after applying an hybrid flat-triangular windowing to the 10-ms weighted ZIR (Processor 17.020). This step uses a 10-ms window $w(n)$ defined below:

$w(n) = 1$ if n=0,..,63,

$w(n) = (128-n)/64$ if n=64,..,127

Note that Processor 17.020 always updates **OVLP_TCX** as **OVLP_TCX** = **ACELP_ZIR**.

The parameter rms_{wsyn} is updated in the ACELP decoder because it is used in the TCX decoder for packet-erasure concealment. Its update in ACELP decoded frames consists of computing per subframe the weighted ACELP synthesis $s_w(n)$ with the perceptual weighting filter (Processor 17.021) and calculating in Processor 17.022 :

$$rms_{wsyn} = \sqrt{\frac{1}{L}\left(s_w(0)^2 + s_w(1)^2 + \ldots + s_w(L-1)^2\right)}$$

where L=256 (20 ms) is the ACELP frame length.

TCX decoding

The TCX decoder is shown in Figure 18. A switch selector (Processor 18.017) is used to handle two different decoding cases:

> **Case 1**: Packet-erasure concealment in TCX-20 (Processors 18.013 to 18.016) when the TCX frame length is 20 ms and the related packet is lost i.e. **BFI_TCX** = (1) ,
>
> **Case 2**: Normal TCX decoding, possibly with partial packet losses (Processors 18.001 to 18.012).

In Case 1, no information is available to decode the 20-ms TCX frame. The TCX synthesis is found by processing the past excitation (Processor 18.013) delayed by *T*, where *T=pitch_tcx* is a pitch lag estimated in the previously decoded TCX frame, by a non-linear filter roughly equivalent to $1/\hat{A}(z)$ (Processors 18.014 to 18.016). A non-linear filter is used instead of $1/\hat{A}(z)$ to avoid clicks in the synthesis. This filter is decomposed in 3 blocks: filtering by $\hat{A}(z/\gamma)/\hat{A}(z)/(1-\alpha z^{-1})$ to map the excitation delayed by T into the TCX target domain (Processor 18.014), limiter (the magnitude is limited to ± rms_{wsyn} in Processor 18.015), and finally filtering by $(1-\alpha\ z^{-1})/\ \hat{A}(z/\gamma)$ to find the synthesis (Processor 18.016). Note that the buffer **OVLP_TCX** is set to zero in this case.

In Case 2, TCX decoding involves decoding the algebraic VQ parameters (Processor 18.002). This decoding step is presented in another part of this detailed description. Recall that the set of transform coefficients $\boldsymbol{Y} = [\ Y_0\ Y_1\ \ldots\ Y_{N-1}\ ]$, where N = 288, 576 and 1152 for TCX-20, 40 and 80 respectively, is divided into K subvectors of dimension 8 which are represented in the lattice RE_8. The number K of subvectors is 36, 72 and 144 for TCX-20, 40 and 80 respectively. Therefore, $\boldsymbol{Y}$ can be expanded as $\boldsymbol{Y} = [\boldsymbol{Y}_0\ \boldsymbol{Y}_1 \ldots.\ \boldsymbol{Y}_{K-1}]$ with $\boldsymbol{Y}_k = [\ Y_{8k} \ldots\ Y_{8k+7}]$ and $k = 0, .., K-1$.

The noise fill-in level σ_{noise} is decoded in Processors 18.003 by inverting the 3-bit uniform scalar quantization used at the encoder. For an index $0 \leq idx_1 \leq 7$, σ_{noise} is given by : $\sigma_{noise} = 0.1 * (8 - idx_1)$. However, it may happen that the index idx_1 is not available. This is the case when **BFI_TCX** = (1) in TCX-20, (1 x) in TCX-40 and (x 1 x x) in TCX-80, with x representing an arbitrary binary value. In this case, σ_{noise} is set to its maximal value, i.e. $\sigma_{noise} = 0.8$.

Comfort noise is injected in the subvectors $\boldsymbol{Y}_k$ rounded to zero and which correspond to a frequency above 6400/6 ≈ 1067 Hz (Processor 18.004). More precisely, $\boldsymbol{Z}$ is initialized as $\boldsymbol{Z} = \boldsymbol{Y}$ and for $K/6 \leq k \leq K$ (only), if $\boldsymbol{Y}_k = (0, 0, \ldots, 0)$, $\boldsymbol{Z}_k$ is replaced by the 8-dimensional vector :

$$\sigma_{noise} * [\ \cos(\theta_1)\ \sin(\theta_1)\ \cos(\theta_2)\ \sin(\theta_2)\ \cos(\theta_3)\ \sin(\theta_3)\ \cos(\theta_4)\ \sin(\theta_4)\ ],$$

where the phases θ_1, θ_2, θ_3 and θ_4 are randomly selected.

The low-frequency deemphasis (Processor 18.005) simply consists of scaling each sub-vector $\boldsymbol{Z}_k$, for $k=0..K/4-1$, by a factor fac_k, which varies with k:

$$X'_k = \mathrm{fac}_k \,.\, \mathbf{Z}_k \ , \ k=0,\ldots,K/4-1.$$

The factor, fac_k, is actually a piecewise-constant monotone-increasing function of k and saturates at 1 for a given $k=k_{max} < K/4$ (i.e. $\mathrm{fac}_k < 1$ for $k < k_{max}$ and $\mathrm{fac}_k = 1$ for $k \geq k_{max}$). The value of k_{max} depends on **Z**. To obtain fac_k, the energy ε_k of each sub-vector $\mathbf{Z}_k$ is computed as follows:

$$\varepsilon_k = \mathbf{Z}_k^{\mathrm{T}} \mathbf{Z}_k + 0.01,$$

where the term 0.01 is set arbitrarily to avoid a zero energy (the inverse of ε_k is later computed). Then, the maximal energy over the first $K/4$ subvectors is searched:

$$\varepsilon_{max} = max(\varepsilon_0, \ldots, \varepsilon_{K/4-1})$$

The actual computation of fac_k is given by the formula below:

$$\mathrm{fac}_0 = \max((\varepsilon_0 / \varepsilon_{max})^{0.5}, 0.1)$$

$$\mathrm{fac}_k = \max((\varepsilon_k / \varepsilon_{max})^{0.5}, \mathrm{fac}_{k-1}) \text{ for } k=1,\ldots, K/4-1$$

The estimation of the dominant pitch (Processor 18.006) is performed so that the next frame to be decoded can be properly extrapolated if it corresponds to TCX-20 and if the related packet is lost. This estimation is based on the assumption that the peak of maximal magnitude in spectrum of the TCX target corresponds to the dominant pitch. The search for the maximum M is restricted to a frequency below 400 Hz

$$M = max_{i=1..N/32} \ (X'_{2i})^2 + (X'_{2i+1})^2$$

and the minimal index $1 \leq i_{max} \leq N/32$ such that $(X'_{2i})^2 + (X'_{2i+1})^2 = M$ is also found. Then the dominant pitch is estimated in number of samples as $T_{est} = N / i_{max}$ (this value may not be integer). Recall that the dominant pitch is calculated for packet-erasure concealment in TCX-20. To avoid buffering problems (the excitation buffer being limited to 20 ms), if $T_{est} > 256$

samples (20 ms), *pitch_tcx* is set to 256 ; otherwise, if $T_{est} \leq 256$, multiple pitch period in 20 ms are avoided by setting *pitch_tcx* to

$$pitch\_tcx = \max\{ \lfloor n\, T_{est} \rfloor \mid n \text{ integer} > 0 \text{ and } n\, T_{est} \leq 256\}$$

where $\lfloor . \rfloor$ denotes the rounding to the nearest integer towards $-\infty$.

The transform used in the illustrative embodiment is a DFT and is implemented as a FFT. Recall that due to the ordering used at the TCX encoder, the transform coefficients $\mathbf{X'}=(X'_0,\ldots,X'_{N-1})$ are such that:

- X'_0 corresponds to the DC coefficient,
- X'_1 corresponds to the Nyquist frequency (i.e. 6400 Hz since the time-domain target signal is sampled at 12.8 kHz), and
- the coefficients X'_{2k} and X'_{2k+1}, for k=1..N/2-1, are the real and imaginary parts of the Fourier component of frequency of $k(/N/2)$ * 6400 Hz.

Processor 18.007 always forces X'_1 to 0. After this zeroing, the time-domain TCX target signal $\boldsymbol{x'}_w$ is found in Processor 18.007 by inverse FFT.

The (global) TCX gain g_{TCX} is decoded in Processor 18.008 by inverting the 7-bit logarithmic quantization used in the TCX encoder. To do so, Processor 18.008 computes the r.m.s. value of the TCX target signal $\boldsymbol{x'}_w$ as:

$$rms = \text{sqrt}(1/N\,(x'^2_{w0} + x'^2_{w1} + \ldots + x'^2_{wL-1}))$$

From an index $0 \leq idx_2 \leq 127$, the TCX gain is given by:

$$g_{TCX} = 10^{\,idx_2/28/(4\times rms)}$$

The (logarithmic) quantization step is around 0.71 dB.

This gain is used in Processor 18.009 to scale $\boldsymbol{x}'_w$ into $\boldsymbol{x}_w$. Note that from the mode extrapolation and the gain repetition strategy used in the illustrative embodiment, the index idx_2 is available to Processor 18.009. However, in case of partial packet losses (1 loss for TCX-40 and up to 2 losses for TCX-80) the least significant bit of idx_2 may be set by default to 0 in the demultiplexer (Processor 18.001).

Since the TCX encoder employs windowing with overlap and weighted ZIR removal prior to transform coding of the target signal, the reconstructed TCX target signal $\boldsymbol{x} = (x_0, x_1, \ldots, x_{N-1})$ is actually found by overlap-add (Processor 18.010). The overlap-add depends on the type of the previous decoded frame (ACELP or TCX). The TCX target signal is first multiplied by an adaptive window $\mathbf{w} = [w_0\ w_1\ \ldots\ w_{N-1}]$:

$$x_i := x_i * w_i, \quad i=0, \ldots, L-1$$

where **w** is defined by

$$w_i = \sin(\pi/ovlp\_len * (i+1)/2), \qquad i = 0, \ldots, ovlp\_len-1$$

$$w_i = 1, \qquad i = ovlp\_len, \ldots, L-1$$

$$w_i = \cos(\pi/(L-N) * (i+1-L)/2), \qquad i = L, \ldots, N-1$$

Note that if $ovlp\_len = 0$, i.e. if the previous decoded frame is ACELP, the left part of this window is skipped. Then, the overlap from the past decoded frame (**OVLP_TCX**) is added to the windowed signal **x** :

$$[\, x_0 \ldots x_{128} \,] := [\, x_0 \ldots x_{128} \,] + \textbf{OVLP\_TCX}$$

If $ovlp\_len = 0$, **OVLP_TCX** is the 10-ms weighted ZIR of ACELP (128 samples) of **x**. Otherwise,

$$\textbf{OVLP\_TCX} = [\, \underbrace{x\ x \ldots}_{olvp\_len \text{ samples}} x\ 0\ 0 \ldots 0 \,],$$

where ovlp_len may be equal to 32, 64 or 128 (2.5, 5 or 10 ms) which indicates that the previously decoded frame is TCX-20, 40 or 80, respectively.

The reconstructed TCX target signal is given by [x_0 ... x_L] and the last *N-L* samples are saved in the buffer **OVLP_TCX** :

$$\mathbf{OVLP\_TCX} := [x_L \ldots x_{N-1}\ \underbrace{0\ 0 \ldots 0}_{128-(L-N)\text{ samples}}]$$

The reconstructed TCX target is filtered (Processor 18.011) by the inverse perceptual filter $W^{-1}(z)=(1-\alpha\ z^{-1})/\hat{A}(z/\gamma)$ to find the synthesis. The excitation is also calculated in Processor 18.012 to update the ACELP adaptive codebook and allow to switch from TCX to ACELP in a subsequent frame. Note that the length of the TCX synthesis is given by the TCX frame length (without the overlap) : 20, 40 or 80 ms.

Decoding of the high-frequency (HF) signal

The decoding of the HF signal implements a kind of bandwidth extension (BWE) mechanism and uses some data from the LF decoder. It is an evolution of the BWE mechanism used in the AMR-WB speech decoder. The HF decoder is detailed in Figure 19. The HF synthesis chain consists of Processors 19.008 to 19.012. More precisely, the HF signal is synthesized in 2 steps: calculation of the HF excitation signal (Processors 19.008 and 19.009), computation of the HF signal from the HF excitation (Processors 19.010 and 19.011). The HF excitation is obtained by shaping in time-domain (Processor 19.008) the LF excitation signal with scalar factors (or gains) per 5-ms subframes. This HF excitation is post-processed in Processor 19.009 to reduce the "buzziness" of the output, and then filtered by a HF linear-predictive synthesis filter $1/A_{HF}(z)$ (Processor 19.010). Recall that the LP order used to encode and then decode the HF signal is 8. The result is also post-processed to smooth energy variations in Processor 19.011.

The HF decoder synthesizes a 80-ms HF superframe. This superframe is segmented according to **MODE** = (m_0, m_1, m_2, m_3). To be more specific, the decoded frames used in the HF decoder are synchronous with the frames used in the LF decoder. Hence, $m_k \leq 1$, $m_k = 2$ and $m_k = 3$ indicate respectively a 20. 40 and 80-ms frame. These frames are referred to as HF-20, HF-40 and HF-80, respectively.

From the synthesis chain described above, it appears that the only parameters needed for HF decoding are ISF and gain parameters. The ISF parameters represent the filter $1/A_{HF}(z)$ (Processor 19.010), while the gain parameters are used to shape the LF excitation signal (Processor 19.008). These parameters are demultiplexed in Processor 19.001 based on **MODE** and knowing the format of the bitstream.

The decoding of the HF parameters is controlled by Processor 15.002. In particular, this processor controls the decoding and interpolation of linear-predictive (LP) parameters (Processors 19.003 and 19.005). It sets proper bad frame indicators to the ISF and gain decoders (Processors 10.003 and 10.007). It also controls the output buffer of the HF signal (Processor 15.005) so that the decoded frames get written in the right time segments of the 80-ms output buffer.

Processor 15.002 generates control data which are internal to the HF decoder : *bfi_isf_hf*, **BFI_GAIN**, the number of subframes for ISF interpolation and a frame selector to set a frame pointer on the output buffer. Except for the frame selector which is self-explanatory, the nature of these data is defined in more details below:

- *bfi_isf_hf* is a binary flag indicating loss of the ISF parameters. Its definition is given below from **BFI** = (bfi_0, bfi_1, bfi_2, bfi_3):

  For HF-20 in packet *k*, *bfi_isf_hf* = bfi_k ,

  For HF-40 in packets *k* and *k+1*, *bfi_isf_hf* = bfi_k ,

  For HF-80 (in packets k=*0 to 3*), *bfi_isf_hf* = bfi_0

  This definition can be readily understood from the bitstream format. Recall that the ISF parameters for the HF signal are always in the first packet describing HF-20, -40 or –80 frames.

- **BFI_GAIN** is a binary vector used to signal packet losses to the HF gain decoder : **BFI_GAIN** = (bfi_k) for HF-20 in packet *k*, (bfi_k bfi_{k+1}) for HF-40 in packets *k* and *k+1*, **BFI_GAIN** = **BFI** for HF-80.

- The number of subframes for ISF interpolation refers to the number of 5-ms subframe in the decoded frame. This number if 4 for HF-20, 8 for HF-40 and 16 for HF-80.

The ISF vector **isf_hf_q** is decoded using AR(1) predictive VQ in Processor 19.003. If *bfi_isf_hf* = 0, the 2-bit index i_1 of the 1st stage and the 7-bit index i_2 of the 2nd stage are available and **isf_hf_q** is given by

$$\mathbf{isf\_hf\_q} = \mathbf{cb1}(i_1) + \mathbf{cb2}(i_2) + \mathbf{mean\_isf\_hf} + \mu_{isf\_hf} * \mathbf{mem\_isf\_hf}$$

where **cb1**(i_1) is the i_1–th codevector of the 1st stage, **cb2**(i_2) is the i_2–th codevector of the 2st stage, **mean_isf_hf** is the mean ISF vector, μ_{isf_hf} = 0.5 is the AR(1) prediction coefficient and **mem_isf_hf** is the memory of the ISF predictive decoder. If *bfi_isf_hf* = 1, the decoded ISF vector corresponds to the previous ISF vector shifted towards the mean ISF vector:

$$\mathbf{isf\_hf\_q} = \alpha_{isf\_hf} * \mathbf{mem\_isf\_hf} + \mathbf{mean\_isf\_hf}$$

with $\alpha_{isf\_hf} = 0.9$. After calculating **isf_hf_q**, the ISF reordering defined in AMR-WB speech coding is applied to **isf_hf_q** with an ISF gap of 180 Hz. Finally the memory **mem_isf_hf** is updated for the next HF frame as:

$$\mathbf{mem\_isf\_hf} = \mathbf{isf\_hf\_q} - \mathbf{mean\_isf\_hf}$$

Note that the initial value of **mem_isf_hf** (at the reset of the decoder) is zero. Processor 19.004 converts the ISF parameters (in frequency domain0 into ISP parameters (in cosine domain).

Processors 19.005 realizes a simple linear interpolation between the ISP parameters of the previous decoded HF frame (HF-20, HF-40 or HF-80) and the new decoded ISP parameters. The interpolation is conducted in the ISF domain and results in ISF parameters for each 5-ms subframe, according to the formula:

$$\mathbf{isp}_{subframe\text{-}i} = i/nb * \mathbf{isp}_{new} + (1\text{-}i/nb) * \mathbf{isp}_{old},$$

where nb is the number of subframes in the current decoded frame (nb=4 for HF-20, 8 for HF-40, 16 for HF-80), i=0,...,nb-1 is the subframe index, $\mathbf{isp}_{old}$ is the set of ISP parameters obtained from the ISF parameters of the previously decoded HF frame and $\mathbf{isp}_{new}$ is the set of ISP parameters obtained from the ISF parameters decoded in Processors 19.003. The interpolated ISP parameters are then converted into linear-predictive coefficients for each subframe in Processor 19.006.

The computation of the gain g_{match} in dB in Processor 19.007 is detailed in the next paragraphs. This gain is interpolated in Processor 19.008 for each 5-ms subframe based on its previous value old_g_{match} as:

$$\tilde{g}_i = i/nb * g_{\text{match}} + (1-i/nb) * old\_g_{\text{match}},$$

where *nb* is the number of subframes in the current decoded frame (*nb*=4 for HF-20, 8 for HF-40, 16 for HF-80), *i*=0,...,*nb*-1 is the subframe index. This results in a vector ($\tilde{g}_0, \ldots \tilde{g}_{nb-1}$).

Gain estimation computation to match magnitude at 6400 Hz (Processor 19.007)

Processor 19.007 is detailed in Figure 11a. Since this process uses only the quantized version of the LPC filters, it is identical to what the encoder has computed at the equivalent stage. A damped sinusoid of frequency 6400 Hz is generated by computing the first 64 samples [$h(0)$ $h(1)$ ... $h(63)$] of the impulse response $h(n)$ of the 1st-order autoregressive filter $1/(1+0.9\ z^{-1})$ having a pole z = -0.9 (Processor 11.017). This 5-ms signal $h(n)$ is passed through the (zero-state) predictor $\hat{A}(z)$ of order 16 whose coefficients are taken from the LF decoder (Processor 11.018), and then the result is passed through the (zero-state) synthesis filter $1/\hat{A}_{HF}(z)$ of order 8 whose coefficients are taken from the HF decoder (Processor 11.019) to obtain the signal $x(n)$. Note that the 2 sets of LP coefficients correspond to the last subframe of the current decoded HF-20, -40 or -80 frame. A correction gain is then computed in dB as $g_{\text{match}} = 10 \log_{10} [\ 1/(x(0)^2 + x(1)^2 + \ \ldots + x(63)^2\ )]$ as illustrated in Processors 11.020.

Recall that the sampling frequency of both the LF and HF signals is 12800 Hz. Furthermore, the LF signal corresponds to the low-passed audio signal, while the HF signal is spectrally a *folded* version of the high-passed audio signal. If the HF signal is a sinusoid at 6400 Hz, it becomes after the synthesis filterbank a sinusoid at 6400 Hz and not 12800 Hz. As a consequence it appears that g_{match} is designed so that the magnitude of the *folded* frequency response of 10^(g_{match}/20) /$A_{HF}(z)$ matches the magnitude of the frequency response of 1/$A(z)$ *around* 6400 Hz.

Decoding of correction gains and gain computation (Processor 19.009)

Recall that after gain interpolation the HF decoder gets from Processor 19.008 the estimated gains (g^{est}_0, g^{est}_1, ..., g^{est}_{nb-1}) in dB for each of the *nb* subframes of the current decoded frame. Furthermore, *nb* = 4, 8 and 16 in HF-20, -40 and –80, respectively. The role of Processor 19.009 is to decode correction gains in dB which will be added to the estimated gains per subframe to form the decode gains $\hat{g}_0, \hat{g}_1, \ldots, \hat{g}_{nb-1}$:

$$(\hat{g}_0(dB), \hat{g}_1(dB), \ldots, \hat{g}_{nb-1}(dB)) = (\tilde{g}_0, \tilde{g}_1, \ldots, \tilde{g}_{nb-1}) + (\bar{g}_0, \bar{g}_1, \ldots, \bar{g}_{nb-1})$$

where

$$(\bar{g}_0, \bar{g}_1, \ldots, \bar{g}_{nb-1}) = (g^{c1}_1, g^{c1}_1, \ldots, g^{c1}_{nb-1}) + (g^{c2}_0, g^{c2}_1, \ldots, g^{c2}_{nb-1}).$$

Therefore, the gain decoding corresponds to the decoding of predictive two-stage VQ-scalar quantization, where the prediction is given by the interpolated 6400 Hz junction matching gain. The quantization dimension is variable and is equal to *nb*.

Decoding of the 1st stage :

The 7-bit index $0 \leq idx \leq 127$ of the 1st stage 4-dimensional HF gain codebook is decoded into 4 gains (G_0, G_1, G_2, G_3). A bad frame indicator *bfi* = *BFI_GAIN$_0$* in HF-20, -40 and –80 allows to handle packet losses. If *bfi* = 0, these gains are decoded as

$$(G_0, G_1, G_2, G_3) = \textbf{cb\_gain\_hf}(\text{idx}) + \textit{mean\_gain\_hf}$$

where **cb_gain_hf**(*idx*) is the *idx*-th codevector of the codebook **cb_gain_hf**. If *bfi* =1, a memory *past_gain_hf_q* is shifted towards –20 dB :

$$past\_gain\_hf\_q := \alpha_{gain\_hf} * (past\_gain\_hf\_q + 20) - 20.$$

where $\alpha_{gain\_hf} = 0.9$ and the 4 gains (G_0, G_1, G_2, G_3) are set to the same value:

$$G_k = past\_gain\_hf\_q + mean\_gain\_hf, \text{ for } k = 0,1,2 \text{ and } 3$$

Then the memory *past_gain_hf_q* is updated as:

$$past\_gain\_hf\_q := (G_0 + G_1 + G_2 + G_3)/4 - mean\_gain\_hf.$$

The computation of the 1st stage reconstruction is then given as:

HF-20: $(g^{c1}_0, g^{c1}_1, g^{c1}_2, g^{c1}_3) = (G_0, G_1, G_2, G_3)$.

HF-40: $(g^{c1}_0, g^{c1}_1, \ldots, g^{c1}_7) = (G_0, G_0, G_1, G_1, G_2, G_2, G_3, G_3)$.

HF-80: $(g^{c1}_0, g^{c1}_1, \ldots, g^{c1}_{15}) = (G_0, G_0, G_0, G_0, G_1, G_1, G_1, G_1, G_2, G_2, G_2, G_2, G_3, G_3, G_3, G_3)$.

Decoding of 2nd stage :

In TCX-20, $(g^{c2}_0, g^{c2}_1, g^{c2}_2, g^{c2}_3)$ is simply set to (0,0,0,0) and there is no real 2nd stage decoding. In HF-40, the 2-bit index $0 \le idx_i \le 3$ of the *i*-th subframe, where *i*=0, ..., 7, is decoded as :

$$\text{If } bfi = 0,\ g^{c2}_i = 3 * idx_i - 4.5 \text{ else } g^{c2}_i = 0.$$

In TCX-80, 16 subframes 3-bit index the $0 \le idx_i \le 7$ of the i-th subframe, where i=0, ..., 15, is decoded as :

$$\text{If } bfi = 0,\ g^{c2}_i = 3 * \text{idx} - 10.5 \text{ else } g^{c2}_i = 0.$$

In TCX-40 the magnitude of the second scalar refinement is up to ± 4.5 dB and in TCX-80 up to ± 10.5 dB. In both cases, the quantization step is 3 dB.

HF gain reconstruction :

The gain for each subframe is then computed in Processor 19.011 as: $10^{\hat{g}_i/20}$

Buzziness reduction (Processor 19.013) and energy smoothing (Processor 19.015)

The role of Processors 19.013 is to attenuate pulses in the time-domain HF excitation signal $r_{HF}(n)$, which often cause the audio output to sound "buzzy". Pulses are detected by checking if the absolute value $| r_{HF}(n) | > 2 * thres(n)$, where $thres(n)$ is an adaptive threshold corresponding to the time-domain envelope of $r_{HF}(n)$. The samples $r_{HF}(n)$ which are detected as pulses are limited to $\pm 2 * thres(n)$, where ± is the sign of $r_{HF}(n)$.

Each sample $r_{HF}(n)$ of the HF excitation is filtered by a 1st order low-pass filter $0.02/(1 - 0.98\ z^{-1})$ to update $thres(n)$. Note that the initial value of $thres(n)$ (at the reset of the decoder) is 0. The amplitude of the pulse attenation is given by :

$$\Delta = \max(\ |r_{HF}(n)| - 2 * thres(n)\ ,\ 0.0).$$

Thus, Δ is set to 0 if the current sample is not detected as a pulse, which will let $r_{HF}(n)$ unchanged. Then, the current value *thres*(*n*) of the adaptive threshold is changed as :

$$thres(n) := thres(n) + 0.5 * \Delta.$$

Finally each sample $r_{HF}(n)$ is modified to : $r'_{HF}(n) = r_{HF}(n) - \Delta$ if $r_{HF}(n) \geq 0$, and $r'_{HF}(n) = r_{HF}(n) + \Delta$ otherwise.

The short-term energy variations of the HF synthesis $s_{HF}(n)$ are smoothed in Processor 19.013. The energy is measured by subframe. The energy of each subframe is modified by up to ± 1.5 dB based on an adaptive threshold.

For a given subframe $[s_{HF}(0)\ s_{HF}(1)\ \ldots\ s_{HF}(63)]$, the subframe energy is calculated as

$$\varepsilon^2 = 0.0001 + s_{HF}(0)^2 + s_{HF}(1)^2 + \ldots + s_{HF}(63)^2.$$

The value *t* of the threshold is updated as:

$$t := \begin{cases} \min(\varepsilon^2 * 1.414, t), & \text{if } \varepsilon^2 < t \\ \max(\varepsilon^2 / 1.414, t), & \text{otherwise.} \end{cases}$$

The current subframe is then scaled by $\sqrt{(t / \varepsilon^2)}$:

$$[s'_{HF}(0)\ s'_{HF}(1)\ \ldots\ s'_{HF}(63)] = \sqrt{(t / \varepsilon^2)} * [s_{HF}(0)\ s_{HF}(1)\ \ldots\ s_{HF}(63)]$$

Post-processing & synthesis filterbank

The post-processing of the LF and HF synthesis and the recombination of the two bands into the original audio bandwidth are illustrated in Figure 21.

The LF synthesis (which is the output of the ACELP/TCX decoder) is first pre-emphasized by the filter (Processor 21.001) of transform function $1/(1-\alpha_{preemph}\ z^{-1})$ where $\alpha_{preemph} = 0.75$. The result is passed through a pitch post-filter (Processor 21.002) to reduce the level coding noise between pitch harmonics only in ACELP decoded segments. This post-filter takes as parameters the pitch gains $\mathbf{g}_p = (g_{p0}, g_{p1}, \ldots, g_{p15})$ and pitch lags $\mathbf{T} = (T_0, T_1, \ldots, T_{15})$ for each 5-ms subframe of the 80-ms superframe. These vectors, $\mathbf{g}_p$ and $\mathbf{T}$ are taken from the ACELP/TCX decoder. Processor 21.003 is the 2nd-order 50 Hz high-pass filter used in AMR-WB speech coding.

The post-processing of the HF synthesis is limited to Processor 21.005, which realizes a simple time alignment of the HF synthesis to make it synchronous with the post-processed LF synthesis. The HF synthesis is thus delayed by 76 samples so as to compensate for the delay incurred by Processor 21.002.

The synthesis filterbank is realized by Processors 21.004, 21.007 and 21.008. The output sampling rate **FS** = 16000 or 24000 Hz is specified as a parameter. The upsampling from 12800 Hz to **FS** in Processors 21.004 and 21.007 is implemented in a similar way as in AMR-WB speech coding. When **FS** = 16000, the LF and HF post-filtered signals are upsampled by 5, processed by a 120-th order FIR filter, then downsampled by 4 and scaled by 5/4. The difference between Processors 21.004 and 21.007 is restricted to the coefficients of the 120-th order FIR filter. Similarly, when **FS** = 24000, the LF and HF post-filtered signals are upsampled by 15, processed by a 368-th order FIR filter, then downsampled by 8 and scaled by 15/8. Processor 21.008 finally combines the two upsampled LF and HF signals to form the 80-ms superframe of the output audio signal.

MULTIPLEXING OF ALGEBRAIC VECTOR QUANTIZATION PARAMETERS INTO ONE OR SEVERAL BINARY TABLES

FOR TCX MODES

Overview

This section discloses how the TCX encoded parameters are put in one or several binary packets for transmission. One packet is used for 20-ms TCX, while respectively 2 and 4 packets are used for 40-ms and 80-ms TCX. To split the TCX spectral information in multiple packets (in case of 40-ms and 80-ms TCX), the spectrum is divided into interleaved *tracks*, where each track contains a subset of the splits in the spectrum (the bits of individual splits are not divided across different tracks). If we number the splits in the spectrum, from low to high frequency, with the split numbers 0, 1, 2, 3, etc. up to the last split at the highest frequency, then the tracks are as follows :

		Split numbers
For 20-ms TCX	Track 1	0, 1, 2, 3, etc. (only one track)
For 40-ms TCX :	Track 1	0, 2, 4, 6, etc.
	Track 2	1, 3, 5, 7, etc.
For 80-ms TCX :	Track 1	0, 4, 8, 12, etc.

Track 2	1, 5, 9, 13, etc.
Track 3	2, 6, 10, 14, etc.
Track 4	3, 7, 11, 15, etc.

Then, recall that the parameters of each split in algebraic VQ consist of the codebook numbers $\mathbf{n} = [n_0 \ldots n_{K-1}]$ and the indices $\mathbf{i} = [i_0 \ldots i_{K-1}]$ of all splits. The values of codebooks numbers **n** are in the set of integers {0, 2, 3, 4,...}. The size (number of bits) of each index i_k is given by $4n_k$. To write these bits into the different packets, we associate a track number to each packet. In the case of 20-ms TCX, only one track is used (i.e. all the splits in the spectrum) and it is written in a single packet. In the case of 40-ms TCX, two packets are used : the first packet is used for Track 1 and the second packet for Track 2. In the case of 80-ms TCX, four packets are used : the first packet is used for Track 1, the second packet for Track 2, the third packet for Track 3 and the fourth packet for Track 4. However, the spectrum quantization and bit allocation was performed without constraining each track to have the same amount of bits, so in general the different tracks do not have the same number of bits allocated to the respective splits. Hence, when writing the encoded splits (codebook numbers and lattice point indices) of a track into their respective packet, two situations can occur : 1) there are not enough bits in the track to fill the packet or 2) there are more bits in a track than the size of the packet so there is overflow. The third possibility (exactly the same number of bits in a track as the packet size) occurs rarely. This overflow has to be managed properly, so all packets are completely filled, and so the decoder can properly interpret and decode the received bits. This overflow management will be explained below when we disclose the multiplexing for the case of multiple binary tables (i.e. tracks).

The split indices are written in their respective packets starting from the lowest frequency split and scanning the track in the spectrum in increasing value of frequency. The codebook number n_K index i_K of each split are written in separate sections of the packet. Specifically, the bits of the codebook number n_K (actually, its unary code representation) are written sequentially starting from one end of the packet, and the bits of the index i_K are written sequentially starting from the other end of the packet. Hence, overflow occurs when these concurrent bit writing processes attempt to overwrite each other. Alternatively, when the bits in one track do not completely fill a packet, there will be a "hole" (i.e. available position for writing more bits) somewhere in the middle of the packet. In 40-ms TCX, overflow will only occur in one of the two packets, while the other packet will have this "hole" where the overflowing bits of the other packet will be written. In 80-ms TCX, there can be "holes" in more than one of the four packets after overflow has happened. In this case, all the "holes" will be grouped together and the overflowing bits of the other packets will be written into these "holes". Details of this procedure are given below.

Then, we note that the use of a unary code to encode the lattice codebook numbers (n) implies that each split requires actually $5n_k$ bits, when it is quantized using a point in the lattice codebook with number n_k. That is, n_k bits are used by the unary code (n_k -1 successive "1's" and a final "0") to indicate how many blocks of 4 bits are used in the codebook index, and $4n_k$ bits are used to form the actual lattice codebook index in codebook n_k) for the split. Note also that when a split is not quantized (i.e. set to zero by the TCX quantizer), it still requires 1 bit (a "0") in the unary code, to indicate that the decoder must skip this split and set it to zero. It is worth noting that, if we do not count the last bit (the "0") of each unary code, then $5n_k$ -1 bits are used by a split quantized with codebook having index n_k. The total number of bits required to index all the quantized splits in the TCX spectrum is thus the sum of the value $5n_k$ -1 for each split (each with possibly different n_k) plus the position of the split (in the TCX spectrum) with highest frequency index that has actually been quantized with a non-zero value (i.e. not set to zero). Note that in this rate consumption calculation, the value $5n_k$ -1 is assumed to be 0 if $n_k = 0$.

Now, more details related to the multiplexing of algebraic vector quantizer indices in one or several packets are given below, in particular regarding the splitting of TCX indices in more than one packet (for 40-ms TCX and 80-ms TCX) and the management of overflow in writing the bits into the packets.

Recall that the codebook numbers are integers defined in the set {0,2,3,4,...., 36}. Each n_k has to be represented in a proper binary format, denoted hereafter $n^E{}_k$, for multiplexing. Unary coding is used in (Ragot, 2002) for this purpose. However, (Ragot, 2002) does not specify any procedure for multiplexing several codebook numbers and indices, i.e. writing all together split encoded codebook numbers $\mathbf{n}^E = [n^E{}_0 \ldots n^E{}_{K-1}]$ and the elements of **i**.

Multiplexing principle for a single binary table

The multiplexing in a single binary table **t** consists of writing bit-by-bit all the elements of **n** and **i** inside **t**, where the table $\mathbf{t} = (t_0, \ldots, t_{R-1})$ contains R bits (which corresponds to the number of bits allocated to algebraic VQ).

A straightforward strategy amounts to writing sequentially the elements of $\mathbf{n}^E$ and **i** in the binary table **t**, as follows:

$$[n^E{}_0 \; i_0 \; n^E{}_1 \; i_1 \; n^E{}_2 \; i_2 \ldots.]$$

In this case, the bits of $n^E{}_0$ are written from position 0 in **t** and upward, the bits of i_0 then follow, etc. This format is uniquely decodable, because the encoded codebook number $n^E{}_k$ indicates the size of i_k.

Instead, in the illustrative embodiment of the invention, an alternative format is used as described below :

$$[\, i_0 \; i_1 \; i_2 \ldots\ldots n^E{}_2 \; n^E{}_1 \; n^E{}_0 \,]$$

The codebook numbers are written sequentially and downward from the end of the binary table **t**, whereas the indices are written sequentially and upward from the beginning of the table. This format has the advantage to separate codebook numbers and indices. This allows to take into account the different bit sensitivity of codebook numbers and indices. Indeed, with the multi-rate lattice vector quantization of (Ragot, 2002) used in the invention, the codebooks numbers are the most sensitive parameters. Since they are written from the beginning of the table **t** and take around 20% of the total bit consumption, they may be protected (e.g. by channel coding) in an efficient and systematic way.

For the actual multiplexing, two pointers are then defined on the binary table **t**: one for (encoded) codebooks numbers pos_n, another for indices pos_i. The pointer pos_i is initialized to 0 (i.e. the beginning of the binary table), and pos_n to R-1 (i.e. the end of the binary table). Positive increments are used for pos_i, and negative ones for pos_n. At any time, the number of bits left in the binary table is given by pos_n-pos_i+1.

The table **t** is initialized to zero. This guarantees that if no data is written, the data inside this table will correspond an all-zero codebook numbers **n** (this follows from the definition of the unary code used here). The splits are then written sequentially in the binary table from $k=0$ to $K-1$: $[n^E_0 \;\; i_0]$ then $[n^E_1 \;\; i_1]$ then $[n^E_2 \;\; i_2]$, etc...

The data of the kth split are really written in the binary table **t** only if the minimal bit consumption of the kth split, denoted R_k hereafter, is less than the number of bits left in table **t**, i.e. if $R_k \leq pos_n - pos_i + 1$. For the multi-rate lattice vector quantization used here, the minimal bit consumption R_k equals to 0 bit if n_k=0, or $5n_k$-1 bits if $n_k \geq 2$.

The multiplexing works as follows:

Initialization*:*

$pos_i = 0$, $pos_n = R-1$

set binary table **t** to zero

For k=0 to K-1 (loop for all splits over the 4 steps below):

1) Compute the number of left bits in table **t**: $nb=pos_n-pos_i+1$
2) Compute the minimal bit consumption of the kth split: $R_k=0$ if n_k=0, $5n_k$-1 if $n_k\geq 2$
3) If $R_k\leq nb$ and $n_k>0$
   a. Write downward $n^E{}_k$ (except the stop bit of the unary code) in table **t** starting from pos_n, and decrement pos_n by n_k -1
   b. Write upward the $4n_k$ bits of i_k from pos_i to pos_i+4n_k-1 in table **t**, and increment pos_i by $4n_k$
   c. Update the number of left bits: $nb := nb-R_k$
4) If $nb\geq 0$, write the stop bit of the unary code and decrement pos_n by 1

In practice, the binary table **t** is physically represented as having 4-bit elements instead of binary (1-bit) elements, so as to accelerate the write-in-table operations and avoid too many bit manipulations. This optimization is significant because the indices i_k are typically formatted into 4-bit blocks. In this case, the value of $\mathrm{pos_i}$ is always a multiple of 4. However, this implies to use bit shifts and modular arithmetic on pointers $\mathrm{pos_n}$ and $\mathrm{pos_i}$ to locate positions in the table.

Moreover, in the multi-rate lattice vector quantization of (Ragot,2002), each index is split into 2 parts: a base codebook number and Voronoi index. This detail does not appear in the above algorithm, but can be easily taken into account by writing i_k in two parts.

Multiplexing (Modules 206 and 207) : case of multiple binary tables

In the case of multiple binary tables, the algebraic VQ parameters are written in P tables $t_0, \ldots, t_{P-1}$ ($P \geq 1$) containing respectively $r_0, \ldots, r_{P-1}$ bits, such that $r_0 + \ldots + r_{P-1} = R$. In other words, the bit budget allocated to algebraic VQ parameters, R, is distributed to P binary tables. In this invention, L is set to 1 in the 20-ms TCX mode, 2 in the 40-ms TCX mode or 4 in the 80-ms TCX mode.

Note that the multiplexing of algebraic VQ parameters in TCX modes employs frame-zero-fill if the bit budget allocated to algebraic VQ is not fully used.

We assume that the number of sub-vectors, K, is a multiple of P. Under this assumption, the algebraic VQ parameters are then divided into P groups of equal cardinality: each group comprises K/P (encoded) codebook numbers and K/P indices. By convention, the pth group is defined as the set $(n^E_{p+jP}, i_{p+jP})_{j=0..K/P-1}$. This can be seen as a decimation operation (in the usual multi-rate signal processing sense). However, another grouping strategy might also be used – for instance, the pth group could also be chosen as $(n^E_{p+lP}, i_{p+lP})_{j=0..K/P-1}$.

Assuming the size of table t_p is sufficient, the parameters of the pth group are written in table t_p. For the sake of clarity, the division of sub-vectors is explained below in more details for P=1 and 2:

- If P=1, the set $(n^E_{p+jP}, i_{p+jP})_{j=0..K/P-1}$ for k=0 simply corresponds to $(n^E_0, i_0, \ldots, n^E_{K-1}, i_{K-1})$. These parameters are written in table t_0. This is the single-table case.
- If P=2, we have $(n^E_{p+jP}, i_{p+jP})_{j=0..K/P-1} = (n^E_0, i_0, n^E_2, i_2 \ldots, n^E_{K-2}, i_{K-2})$ for p=0 and $(n^E_1, i_1, n^E_3, i_3 \ldots, n^E_{K-1}, i_{K-1})$ for p=1. Assuming the table sizes are sufficient, the parameters $(n^E_0, i_0, n^E_2, i_2 \ldots, n^E_{K-2}, i_{K-2})$ are

written in table $\mathbf{t}_0$, while the other parameters (n^E_1, i_1, n^E_3, i_3..., n^E_{K-1}, i_{K-1}) are written in table $\mathbf{t}_1$.

The case of P=4 can be readily understood from the case of P=2.

As a consequence, in principle the multiplexing in the multiple-table case boils down to applying several times the single-table multiplexing principle: the (encoded) codebook numbers $(n^E_{p+jP})_{j=0..K/P-1}$ can be written upward from the bottom of each table $\mathbf{t}_p$ and the indices $(i_{p+jP})_{j=0..K/P-1}$ can be written downward from the end of each table $\mathbf{t}_p$. Two pointers are defined for each binary table $\mathbf{t}_p$: $pos_{n,p}$ and $pos_{i,p}$. These pointers are initialized to $pos_{i,p} = 0$ and $pos_{n,p} = r_p - 1$, and are respectively incremented and decremented.

Nonetheless, the multiple-table case is not a straightforward extension of the single-packet case. It may happen indeed that the number of bits in $(n^E_{p+jP}, i_{p+jP})_{j=0..K/P-1}$ exceeds, for a given p, the number of bits, r_p, available in the binary table $\mathbf{t}_p$. To deal with such an "overflow", an extra table $\mathbf{t}_{ex}$ is defined as temporary buffer to write the bits in excess (which have to be distributed in another table $\mathbf{t}_q$ with $q \neq p$). The size of $\mathbf{t}_{ex}$ is set to 4*36 bits. This size can be justified by the following arguments:

- In the illustrative embodiment of this invention the bits in excess always correspond to a specific index index i_k (not encoded codebook numbers).
- The size of an index i_k is $4n_k$ bits, and the maximum codebook number n_k is 36, hence a maximum size for i_k of 4*36 bits.

The actual multiplexing algorithm in the multiple-table case is detailed below :

Initialize:

We assume that a size of r_p bits for each binary table $\mathbf{t}_p$.

```
Set total number of bits to $R$: $nb = R$

Initialize the maximum position $last$ such that $n_{last} \geq 2$ :

    $last = -1$

For $p$=0...$P$-1,

    $pos_{l,p} = 0$ and $pos_{n,p} = r_p - 1$,

    set table $\mathbf{t}_p$ to zero
```

Split and write all codebook numbers:

For p=0...P-1, the (encoded) codebook numbers $(n^E_{p+jP})_{j=0..K/P-1}$ are written sequentially (downward from the end) in table $\mathbf{t}_p$. This is done through two nested loops over p and j. In the illustrative embodiment a single loop is used with modular arithmetic, as detailed below:

```
For $k$=0,...,$K$-1

    $p = k \bmod P$

    Compute the minimal bit consumption of the $k$th split: $R_k = 0$ if $n_k$=0,
    $5n_k$-1 if $n_k \geq 2$

    If $R_k > nb$, $n_k$=0 else $nb = nb - R_k$

    If $n_k \geq 2$, $last = k$
```

Write downward n^E_k (except the stop bit of the unary code) in table $\mathbf{t}_p$ starting from $pos_{n,p}$, and decrement $pos_{n,p}$ by n_k -1

If $nb \geq 0$, write the stop bit of the unary code and decrement $pos_{n,p}$ by 1

It can be checked that with the conditions of the illustrative embodiment (in particular $P \leq 4$ with a near-equal distribution of R in r_p), no overflow (i.e. bit in excess) in tables $\mathbf{t}_l$ can happen at this step (for p=0,..,P-1). In general this property must be verified to apply the algorithm.

Split and write all indices:

This is the tricky part of the multiplexing algorithm due to the possibility of overflow.

Find the positions pos^{ovf}_p in each binary table $\mathbf{t}_p$ (with p = 1…P) from which the bits in overflow can be written. These positions are computed assuming the indices are written by 4-bit block.

```
For $p = 0..P-1$
        $pos = 0$
        $nb = pos_{n,p} + 1$
        For $k = p$ to $last$ with a step of $P$
                If $n_k > 0$,
                        If $4n_k \leq nb$, $nb_1 = n_k$
                        else $nb_1 = nb >> 2$ (where >> is a bit shift operator)
        $nb = nb - 4^{*} nb_1$
        $pos = pos + nb_1$

    $pos^{ovf}_p = pos^{*}4$
```

The indices can then be written as follows:

```
For $p = 0..P-1$

    $pos = 0$

    For $l = p$ to $N-1$ with a step of $P$

        $nb = pos_{n,p} - pos$

        *Write the $4n_k$ bits of $i_k$:*

        Compute the number, $nb_1$, of 4-bit blocks which can fit in
        table $\mathbf{t}_p$ and the number, $nb_2$, of 4-bit blocks in excess (to be
        written temporarily in table $\mathbf{t}_{ex}$):

            If $4n_k \le nb$, $nb_1 = n_k$, $nb_2 = 0$

            else $nb_1 = nb >> 2$ (where >> is a bit shift operator),
            $nb_2 = n_k - nb_1$

        Write upward the $4nb_1$ bits of $i_k$ from $pos_{i,p}$ to $pos_{i,p}+4nb_1-1$ in
        table $\mathbf{t}_p$, and increment $pos_{i,p}$ by $4nb_1$

        If $nb_2 \ge 0$,

            Initialize $pos_{ovf}$ to 0
```

Write upward the remaining $4nb_2$ bits of i_k from pos_{ovf} to $pos_{ovf}+4nb_2-1$ in table $\mathbf{t}_{ex}$, and increment pos_{ovf} by $4nb_{ovf}$

Distribute the $4nb_2$ bits in table $\mathbf{t}_p$ (with $q \neq p$) based on the pointers $pos^{ovf}{}_q$ and $pos_{n,q}$ and the pointers $pos^{ovf}{}_q$ are updated

FORMATTING OF THE ACELP/TCX BITSTREAM (PACKETIZATION)

Packetization Procedure

In the illustrative embodiment, the coding parameters computed in a 80-ms super-frame at the encoder are multiplexed into 4 binary packets of equal size. The packetization consists of a multiplexing loop over 4 iterations ; the size of each packet is set to $R_{total} / 4$ where R_{total} is the number of bits allocated to the super-frame.

Recall that the mode selected in the 80-ms super-frame has the form (m_1, m_2, m_3, m_4), where m_k=0, 1, 2 or 3, with the mapping

0 → 20-ms ACELP

1 → 20-ms TCX

2 → 40-ms TCX

3 → 80-ms TCX

The multiplexing in the *k*-th packet is performed according to the value of m_k. The corresponding packet format is shown in Figure 3. There are 3 cases:

- If m_k=0 or 1, the *k*-th packet simply contains all parameters related to a 20-ms frame, where are the 2-bit mode information ('00' or '01' in binary format), the parameters of ACELP or those of 20-ms TCX, and the parameters of 20-ms HF coding.

- If m_k=2, the *p*-th packet contains half of the bits of the 40-ms TCX mode, half of the bits of 40-ms HF coding, plus the 2-bit mode information ('10' in binary format).

- If m_k=3, the *k*-th packet contains one fourth of the bits describing the 40-ms TCX mode, one fourth of the bits of 80-ms HF coding, plus the 2-bit mode information ('11' in binary format).

The packetization is therefore straightforward if the *k*-th packet corresponds to ACELP or 20-ms TCX. The packetization is slightly more involved if 40- or 80-ms TCX mode is used, because the bits of the 40- or 80-ms modes have to be shared into even parts.

Bitstream format

The actual bitstream simply consists in a succession of 20-ms binary packets, with a synchronization word preceding each packet, as shown in Fig. 3 (where the synchronization word is not shown).

The bit rate is fixed at the encoder, therefore the packet size is also fixed (equal to *R/4* where *R* is the total bit allocation per 80-ms super-frame).

Each 20-ms packet is written sequentially bit-by-bit in the bitstream. A synchronization word is typically defined at the beginning of each packet.

TCX GAIN ENCODING AND MULTIPLEXING

It was found that the TCX gain is important to maintain audible quality. Thus, in 40-ms and 80-ms TCX frames, the TCX gain value is encoded redundantly in multiple packets to protect against packet loss. The TCX gain is encoded at a resolution of 7 bits, and these bits are labeled "Bit 0" to "Bit 6", where "Bit 0" is the Least Significant Bit (LSB) and "Bit 6" is the Most Significant Bit (MSB). We consider two cases, TCX40 and TCX80, where the encoded bits are split into two or four packets, respectively.

At the Encoder side

TCX40: The first packet contains the full gain information (7 bits). The second packet repeats the most significant 6 bits ("Bit 1" to "Bit 7").

TCX80: The first packet contains the full gain information (7 bits). The third packet contains a copy of the three bits "Bit 4", "Bit 5" and "Bit 6". The fourth packet contains a copy of the three bits "Bit 1", "Bit 2" and "Bit 3".

Additionally, a 3-bit "parity" is formed as thus: combining by logical XOR "Bit 1" and "Bit 4" to generate "Parity Bit 0", combining by logical XOR "Bit 2" and "Bit 5" to generate "Parity Bit 1", and combining by logical XOR "Bit 3" and "Bit 6" to generate "Parity Bit 2". These three parity bits are sent in the second packet.

At the Decoder side

The logic applied at the decoder to recover the TCX gain when missing packets occur for 40-ms TCX and 80-ms TCX is shown in the flowchart of

Figure 22. We assume that there is at least one packet missing before entering the flowchart.

TCX40: If the fist packet is flagged as being lost, the TCX global gain is taken from the second packet, with the LSB ("Bit 0") being set to zero. If only the second packet is lost, then the full TCX gain is obtained from the first packet.

TCX80: The gain recovery algorithm is only used if 1 or 2 packets forming an 80-ms TCX frame are lost; as described in the Mode Extrapolation section of the detailed description of the decoder, if 3 or more packets are lost in a TCX80 frame, the MODE is changed to (1,1,1,1) and BFI=(1,1,1,1). When only 1 or 2 packets are lost in a TCX80 frame, the recovery algorithm is as follows (see Figure 22):

As described above, the second, third and fourth packets of a TCX80 frame contain the parity bits, "Bit 6" to "Bit 4", and "Bit 3" to "Bit 1" of the TCX gain. These bits (three each) are stored in "parity", "index0" and "index1" respectively (Processor 22.004).

If the third packet is lost, "index0" is replaced by the logical XOR combination of "parity" and "index1" (Processors 22.005 and 22.006). That is, "Bit 6" is generated from the logical XOR of "Parity Bit 2" and "Bit 3", "Bit 5" is generated from the logical XOR of "Parity Bit 1" and "Bit 2", and "Bit 4" is generated from the logical XOR of "Parity Bit 0" and "Bit 1".

If the fourth packet is lost, "index1" is replaced by the logical XOR combination of "parity" and "index0" (Processors 22.007 and 22.008). That is, "Bit 3" is generated from the logical XOR of "Parity Bit 2" and "Bit 6", "Bit 2" is generated from the logical XOR of "Parity Bit 1" and "Bit 5", and "Bit 1" is generated from the logical XOR of "Parity Bit 0" and "Bit 4".

Finally, the 7-bit TCX gain value is taken from the recovered bits ("Bit 1" to "Bit 6") and "Bit 0" is set to zero (Processor 22.009).

Table A-1

List of the key symbols in accordance with the illustrative embodiment of the invention

(a) self-scalable multirate RE_8 vector quantization.

Symbol	Meaning	Note
N	dimension of vector quantization	
Λ	(regular) lattice in dimension N	
RE_8	Gosset lattice in dimension 8.	
x or X	Source vector in dimension 8.	
y or Y	Closest lattice point to $\underline{x}$ in RE_8.	
n	Codebook number, restricted to the set $\{0, 2, 3, 4, 5, \ldots\}$.	
Q_n	Lattice codebook in $\Lambda$of index n.	In the self-scalable multirate RE_8 vector quantizer, Q_n is indexed with $4n$ bits.
i	Index of the lattice pointy in a codebook Q_n.	In the self-scalable multirate RE_8 vector quantizer, the index i is represented with $4n$ bits.

n_E	Binary representation of the codebook number n	See Table 2 for an example.
R	bit allocation to self-scalable multirate RE_8 vector quantization (i.e. available bit budget to quantize x)	

(b) split self-scalable multirate RE_8 vector quantization.

Symbol	Meaning	Note
$\lceil . \rceil$	rounding to the nearest integer towards $+\infty$	sometimes called ceil()
N	dimension of vector quantization	multiple of 8
K	number of 8-dimensional subvectors	$N=8K$
RE_8	Gosset lattice in dimension 8.	
RE_8^K	cartesian product of RE_8 (K times): $RE_8^K = RE_8 \otimes \ldots \otimes RE_8$	this is a N-dimensional lattice
$\underline{z}$	N-dimensional source vector	
$\underline{x}$	N-dimensional input vector for split RE_8 vector quantization	$\underline{x}=1/g\ \underline{z}$
g	gain parameter of gain-shape vector quantization	
$\underline{e}$	vector of split energies (K-tuple)	$\underline{e}=(e(0), \ldots ,e(K-1))$ $e(k) = z(8k)^2 + \ldots + z(8k+7)^2,\ 0 \le k \le K-1$
$\underline{R}$	vector of estimated split bit budget (K-tuple) for $g=1$	$\underline{R}=(R(0), \ldots ,R(K-1))$

$\underline{b}$	vector of estimated split bit allocations (K-tuple) for a given *offset*	$\underline{b}=(b(0), \ldots, b(K-1))$ for a given *offset*, $b(k) = R(k) - offset$, if $b(k) < 0$, $b(k) := 0$
offset	integer offset in logarithmic domain used in the discrete search for the optimal g	$g=2^{offset/10}$ $0 \leq offset \leq 255$
fac	noise level estimate	
$\underline{y}$	closest lattice point to $\underline{x}$ in RE_8^K	
$\underline{nq}$	vector of codebook numbers (K-tuple)	$\underline{nq}=(nq(0), \ldots, nq(K-1)_1)$ each entry $nq(k)$ is restricted to the set $\{0, 2, 3, 4, 5, \ldots\}$.
Q_n	Lattice codebook in RE_8 of index n.	Q_n is indexed with $4n$ bits.
$\underline{iq}$	vector of indices (K-tuple)	$\underline{iq}=(iq(0), \ldots, iq(K-1))$ the index $iq(k)$ is represented with $4nq(k)$ bits.
$\underline{nq_E}$	vector of (variable-length) binary representations for the codebook numbers in $\underline{nq'}$	See Table 2 for an example.
R	bit allocation to split self-scalable multirate RE_8 vector quantization (i.e.	

	available bit budget to quantize x)	
<u>*nq'*</u>	vector of codebook numbers (*K*-tuple) such that the bit budget necessary to multiplex of <u>*nq*$_E$</u> and <u>*iq*</u> (until subvecotr *last*) does not exceed *R*	<u>*nq'*</u>$=(nq'(0), \ldots, nq'(K-1))$ each entry $nq'(k)_{()}$ is restricted to the set $\{0, 2, 3, 4, 5, \ldots\}$.
last	index of the last subvector to be multiplexed in formatting table *parm*	$0 \leq last \leq K-1$
<u>*pos*</u>	indices of subvectors sorted with respect to their split energies	<u>*pos*</u>$=(ps(0), \ldots, pos(K-1)_{l})$ <u>*pos*</u> is a permutation of $(0,1,\ldots,K-1)$ $e(pos(0)) \geq e(pos((1)) \geq \ldots \geq e(pos(K-1))$
parm	integer formatting table for multiplexing	$\lceil R/4 \rceil$ integer entries each entry has 4 bits, except for the last one which has (R *mod* 4) bits if *R* is not a multiple of 4, otherwise 4 bits.
pos_i	pointer to write/read indices in formatting table *parm*	in the single-packet case: initialized to 0, incremented by integer steps multiple of 4
pos_n	pointer to write/read codebook numbers in formatting table *parm*	in the single-packet case: initialized to *R-1*, decremented by integer steps

(c) transform coding based on split self-scalable multirate RE_8 vector quantization.

Symbol	Meaning	Note
N	dimension of vector quantization	
RE_8 R	Gosset lattice in dimension 8. bit allocation to self-scalable multirate RE_8 vector quantization (i.e. available bit budget to quantize x)	

REFERENCES

(Jayant, 1984)	N.S. Jayant and P. Noll, *Digital Coding of Waveforms - Principles and Applications to Speech and Video*, Prentice-Hall, 1984
(Gersho, 1992)	A. Gersho and R.M. Gray, *Vector quantization and signal compression*, Kluwer Academic Publishers, 1992
(Kleijn, 1995)	W.B. Kleijn and K.P. Paliwal, *Speech coding and synthesis*, Elsevier, 1995
(Gibson, 1988)	J.D. Gibson and K. Sayood, "*Lattice Quantization*," Adv. Electron. Phys., vol. 72, pp. 259-331, 1988
(Lefebvre, 1994)	R. Lefebvre and R. Salami and C. Laflamme and J.-P. Adoul, "*High quality coding of wideband audio signals using transform coded excitation (TCX)*," Proceedings IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP), vol. 1, 19-22 April 1994, pp. I/193 - I/196
(Xie, 1996)	M. Xie and J-P. Adoul, "*Embedded algebraic vector quantizers (EAVQ) with application to wideband speech coding*," Proceedings IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP), vol. 1 , 7-10 May 1996 , pp. 240 -243
(Ragot, 2002)	S. Ragot, B. Bessette and J.-P. Adoul, *A Method and System for Multi-Rate Lattice Vector Quantization of a Signal*, Canadian patent 2 388 358, 31 May 02
(Jbira, 1998)	A. Jbira and N. Moreau and P. Dymarski, "*Low delay coding of wideband audio (20 Hz-15 kHz) at 64 kbps*," Proceedings IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP), vol. 6, 12-15 May 1998, pp. 3645 -3648

(Schnitzler, 1999)	J. Schnitzler et al., "*Wideband speech coding using forward/backward adaptive prediction with mixed time/frequency domain excitation,*" Proceedings IEEE Workshop on Speech Coding Proceedings, 20-23 June 1999, pp. 4-6
(Moreau, 1992)	N. Moreau and P. Dymarski, "*Successive orthogonalizations in the multistage CELP coder,*" Proceedings IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP), 1992, pp. 61-64
(Bessette, 2002)	B. Bessette et al., "*The adaptive multirate wideband speech codec (AMR-WB),*" *IEEE Transactions on Speech and Audio Processing*, vol. 10, no. 8, Nov. 2002, pp. 620 -636
(Bessette, 1999)	B. Bessette and R. Salami and C. Laflamme and R. Lefebvre, "*A wideband speech and audio codec at 16/24/32 kbit/s using hybrid ACELP/TCX techniques,*" Proceedings IEEE Workshop on Speech Coding Proceedings, 20-23 June 1999, pp. 7-9
(Chen, 1997)	J.-H. Chen, "*A candidate coder for the ITU-T's new wideband speech coding standard,*" Proceedings IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP), vol. 2 , 21-24 April 1997, pp. 1359-1362
(Chen, 1996)	J.-H. Chen and D. Wang, "*Transform predictive coding of wideband speech signals,*" Proceedings IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP), vol. 1, 7-10 May 1996, pp. 275-278
(Ramprashad, 2001)	S.A. Ramprashad, "The multimode transform predictive coding paradigm," *IEEE Transactions on Speech and Audio Processing*, vol. 11, no. 2 , March 2003, pp. 117-129
(Combescure, 1999)	P. Combescure et al., "*A 16, 24, 32 kbit/s wideband speech codec based on ATCELP,*" Proceedings IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP), vol. 1, 15-19 March 1999, pp. 5-8

(3GPP TS 26.190)	3GPP TS 26.190, "AMR Wideband Speech Codec; Transcoding Functions".
(3GPP TS 26.173)	3GPP TS 26.173, "ANSI-C code for AMR Wideband speech codec".

Parameter	Bit Allocation per 20-ms Frame				
	13.6k	16.8k	19.2k	20.8k	24k
ISF Parameters	46				
Mean Energy	2				
Pitch Lag	32				
Pitch Filter	4 × 1				

Parameter	Bit Allocation per 20-ms Frame				
	13.6k	16.8k	19.2k	20.8k	24k
ISF Parameters	46				
Mean Energy	2				
Pitch Lag	32				
Pitch Filter	4 × 1				
Fixed-codebook Indices	4 × 36	4 × 52	4 × 64	4 × 72	4 × 88
Codebook Gains	4 × 7				
Total in bits	254	318	366	398	462

Table 4. Bit allocation of the ACELP frame per 20 ms .

Parameter	Bit allocation per 20-ms frame				
	13.6k	16.8k	19.2k	20.8k	24k
ISF Parameters	46				
Noise Factor	3				
Global Gain	7				
Algebraic VQ	198	262	310	342	406
Total in bits	254	318	366	398	462

Table 5a. Bit allocation of the 20-ms TCX frames .

Parameter	Bit allocation per 40-ms frame (1st 20-ms frame, 2nd 20-ms frame)				
	13.6k	16.8k	19.2k	20.8k	24k
ISF Parameters	46 (16,30)				
Noise Factor	3 (3,0)				
Global Gain	13 (7,6)				
Algebraic VQ	446 (228,218)	574 (292,282)	670 (340,330)	734 (372,362)	862 (436,426)
Total in bits	508	636	732	796	924

Table 5b. Bit allocation of the 40-ms TCX frames .

Parameter	Bit allocation per 80-ms frame (1st, 2nd, 3rd, 4th 20-ms frame)				
	13.6k	16.8k	19.2k	20.8k	24k
ISF Parameters	46 (16,6,12,12)				
Noise Factor	3 (0,3,0,0)				
Global Gain	16 (7,3,3,3)				
Algebraic VQ	960 (231,242,239,239)	1207 (295,306,303,303)	1399 (343,354,359,359)	1536 (375,386,383,383)	1792 (439,450,447,447)
Total in bits	1016	1272	1464	1592	1848

Table c. Bit allocation of the 80-ms TCX frames .

Parameter	Bit allocation per 20 / 40 / 80-ms frame
ISF Parameters	9 (2 + 7)
Gain	7
Gain Corrections	0 / 8 × 2 / 16 × 3
Total in bits	16 / 32 / 64

Table 6. Bit allocation of the bandwidth extension

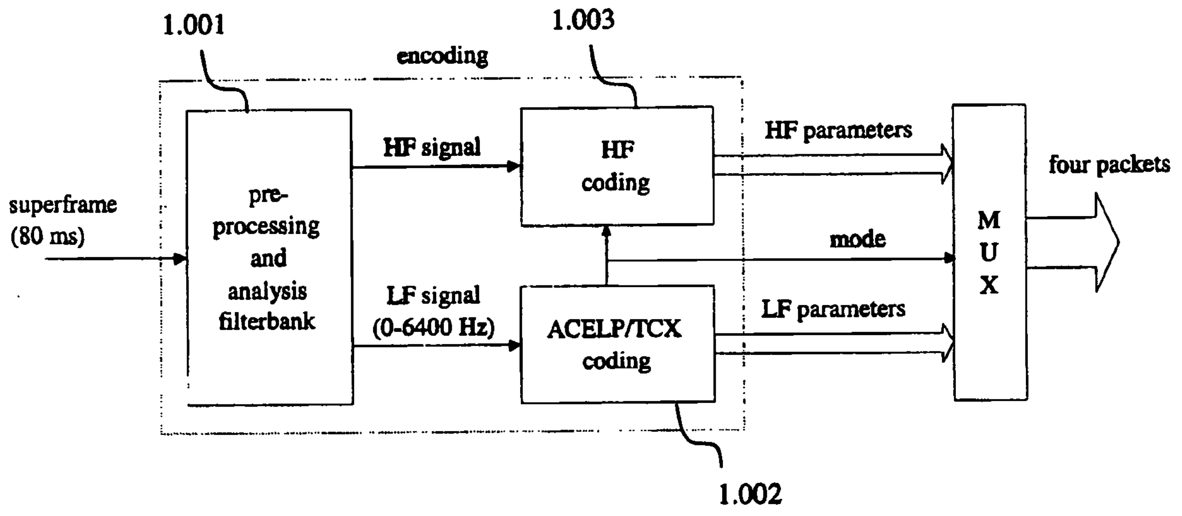


Figure 1. High-level description of the encoder

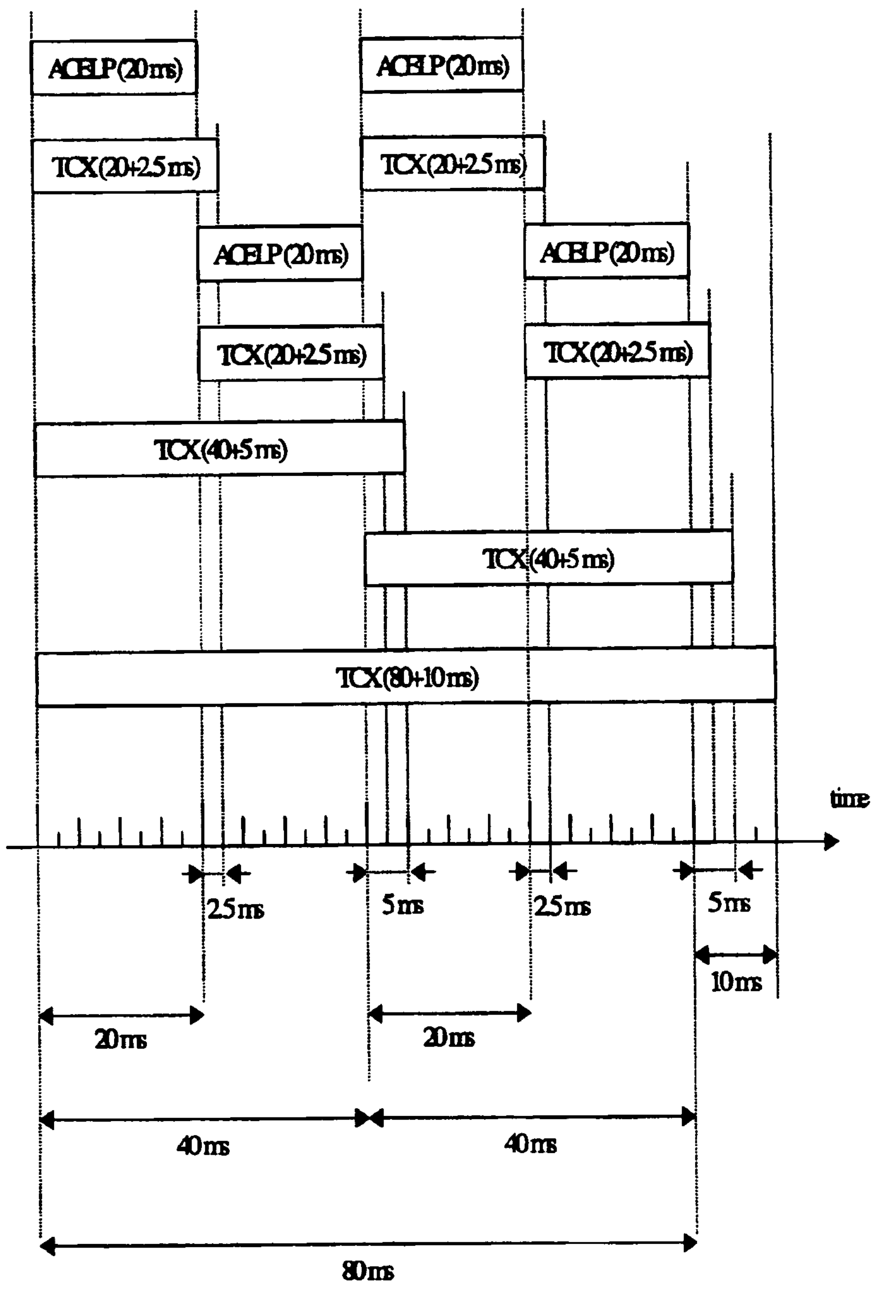


Figure 2. Timing chart of the frame types.

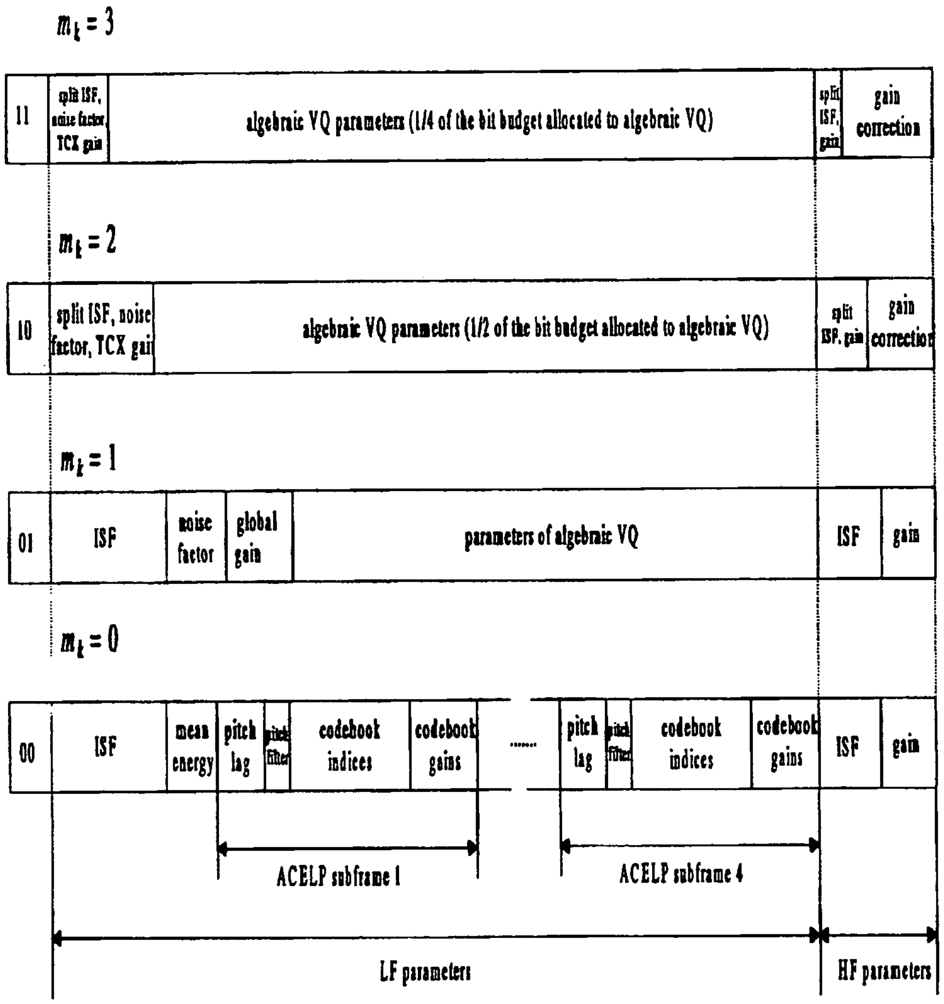


Figure 3. Structure of the payload for all four frame types.

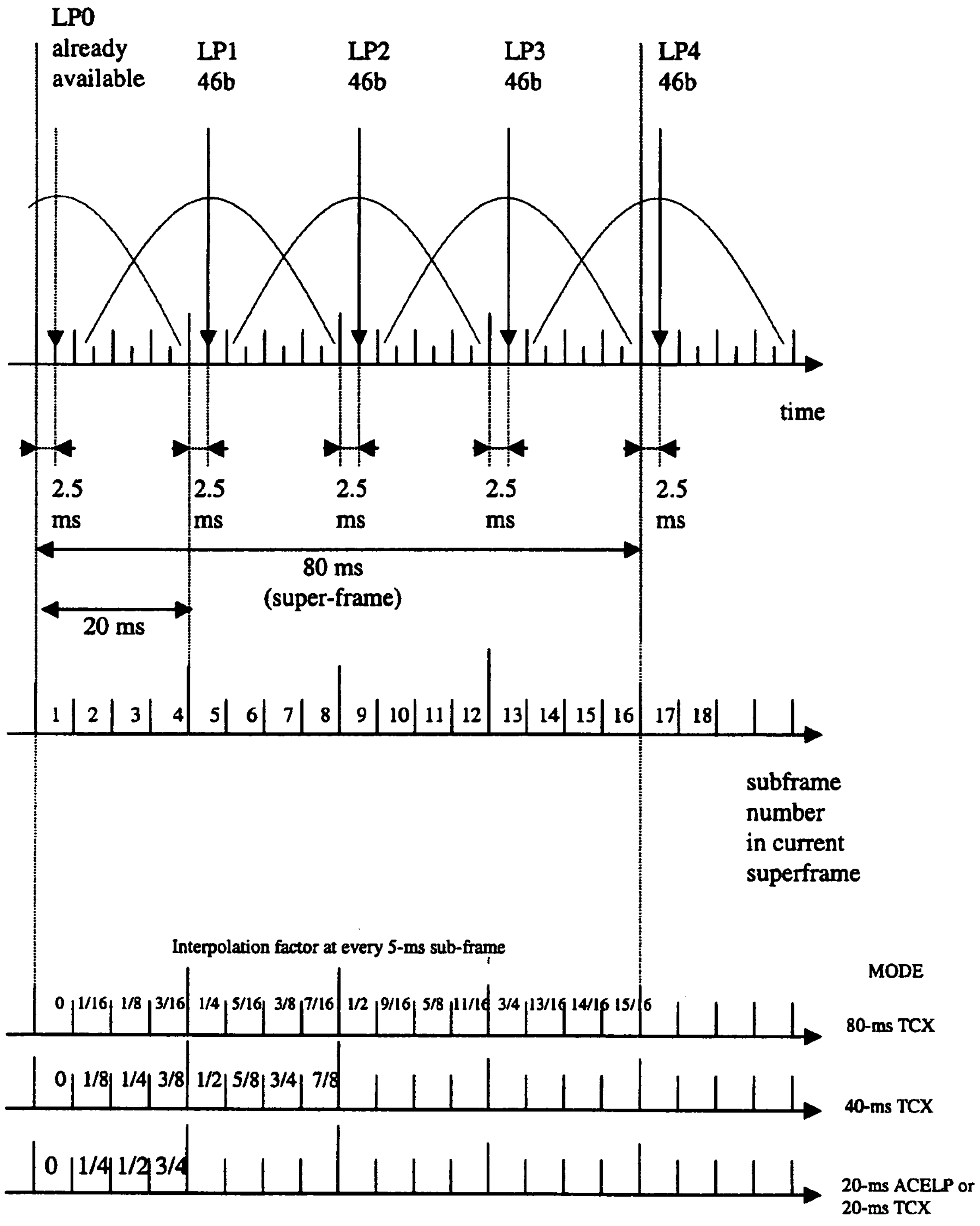
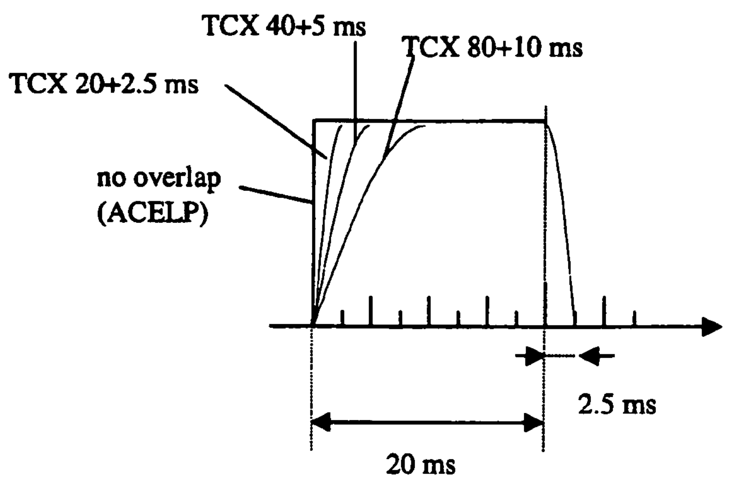
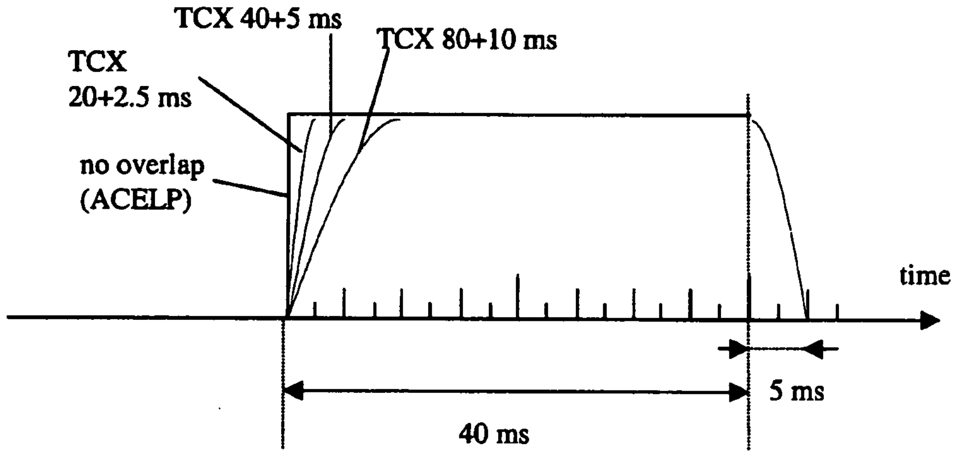


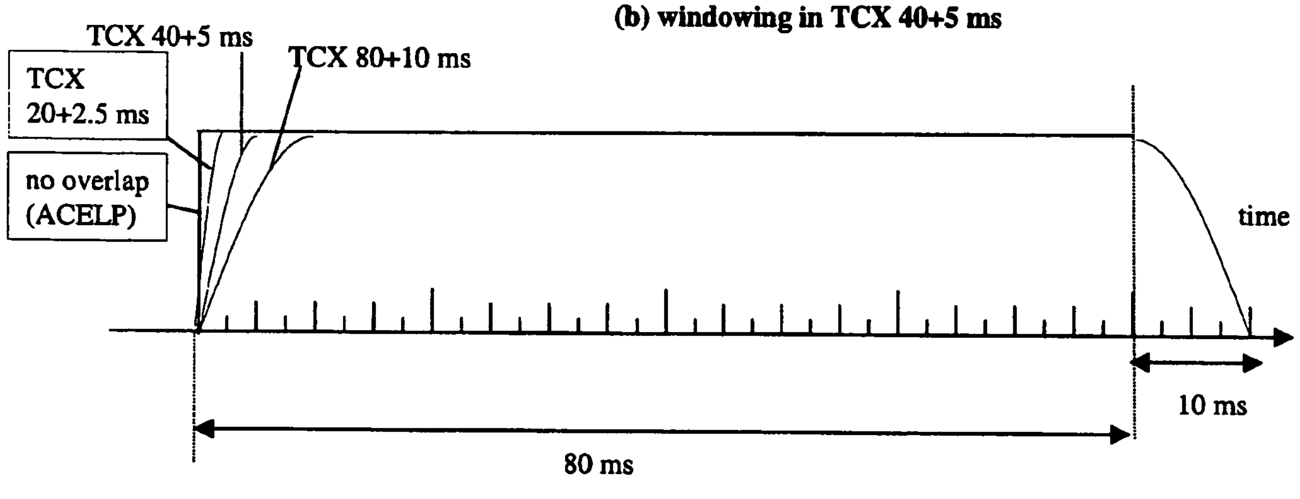
Figure 4. Windowing for linear predictive analysis and interpolation.



(a) windowing in TCX 20+2.5 ms

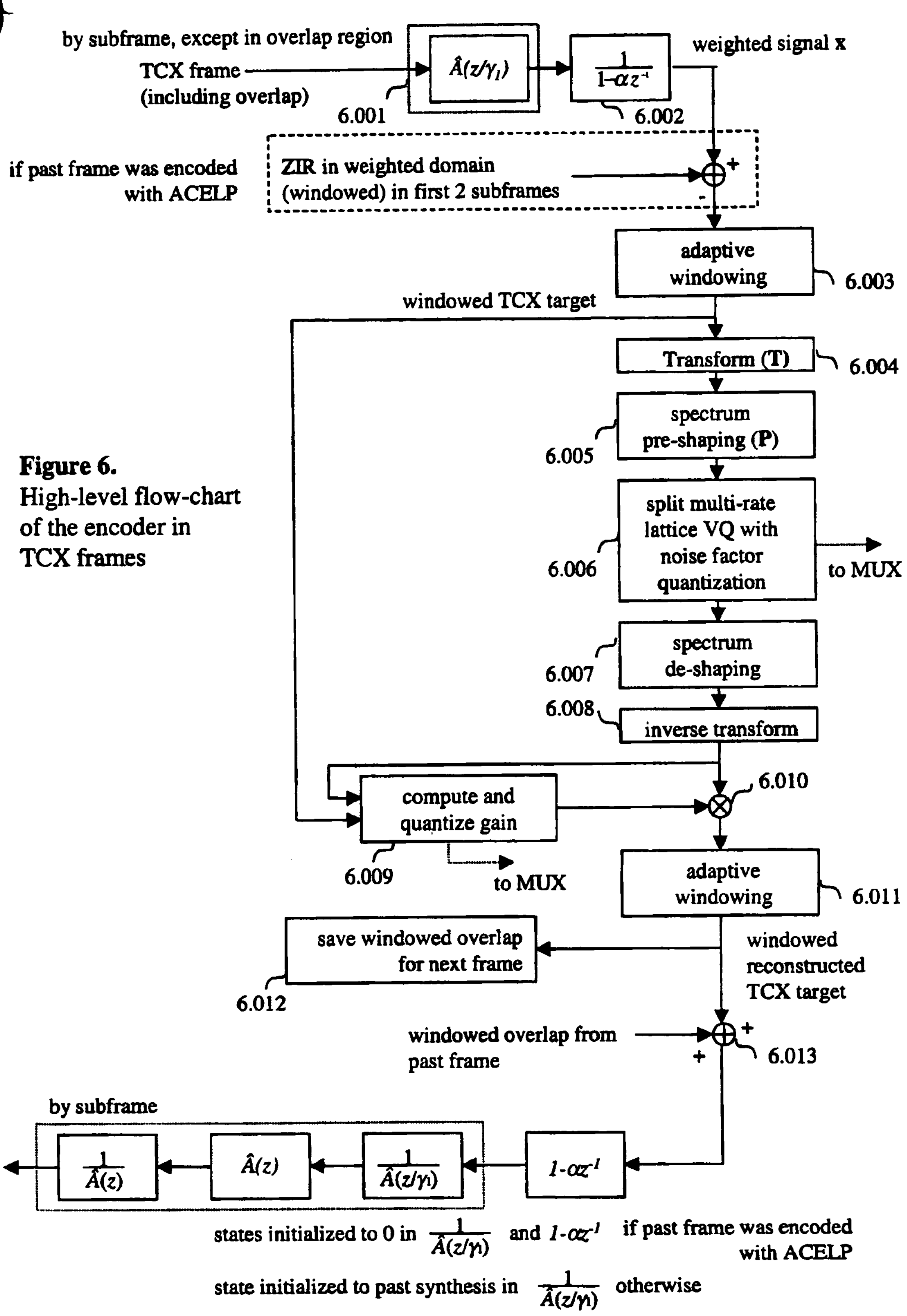


(b) windowing in TCX 40+5 ms



(c) in TCX 80+10 ms

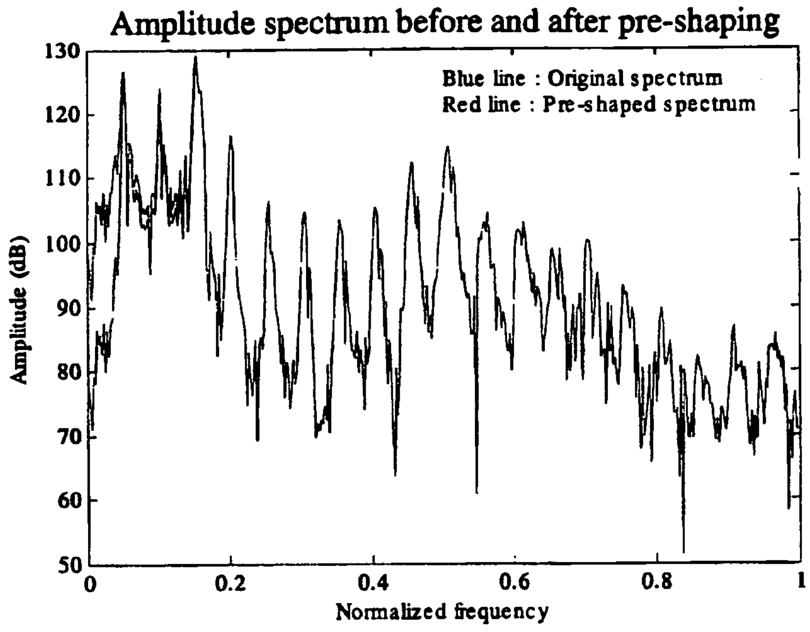
Figure 5. Frame windowing in ACELP/TCX encoder..



by subframe, except in overlap region
TCX frame (including overlap)
$\hat{A}(z/\gamma_1)$
$\frac{1}{1-\alpha z^{-1}}$
weighted signal x
6.001
6.002
if past frame was encoded with ACELP
ZIR in weighted domain (windowed) in first 2 subframes
+
-
adaptive windowing
6.003
windowed TCX target
Transform (T)
6.004
spectrum pre-shaping (P)
6.005
split multi-rate lattice VQ with noise factor quantization
6.006
to MUX
spectrum de-shaping
6.007
6.008
inverse transform
compute and quantize gain
6.009
to MUX
6.010
adaptive windowing
6.011
save windowed overlap for next frame
6.012
windowed reconstructed TCX target
windowed overlap from past frame
+
+
6.013
by subframe
$\frac{1}{\hat{A}(z)}$
$\hat{A}(z)$
$\frac{1}{\hat{A}(z/\gamma_1)}$
$1-\alpha z^{-1}$
states initialized to 0 in $\frac{1}{\hat{A}(z/\gamma_1)}$ and $1-\alpha z^{-1}$ if past frame was encoded with ACELP
state initialized to past synthesis in $\frac{1}{\hat{A}(z/\gamma_1)}$ otherwise

Figure 6.
High-level flow-chart of the encoder in TCX frames

(a)



(b)

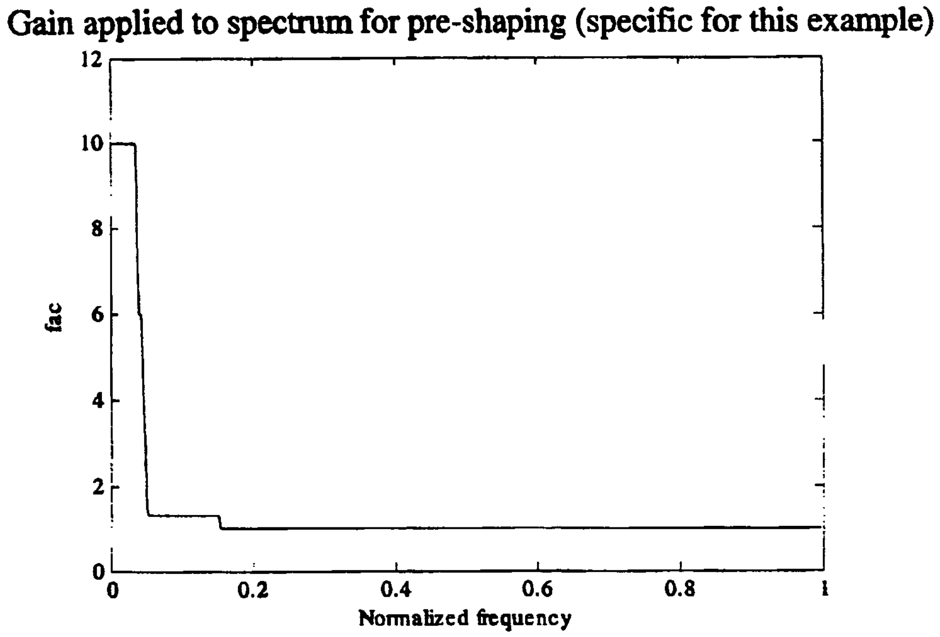


Fig 6a. Example of a pre-shaped spectrum.

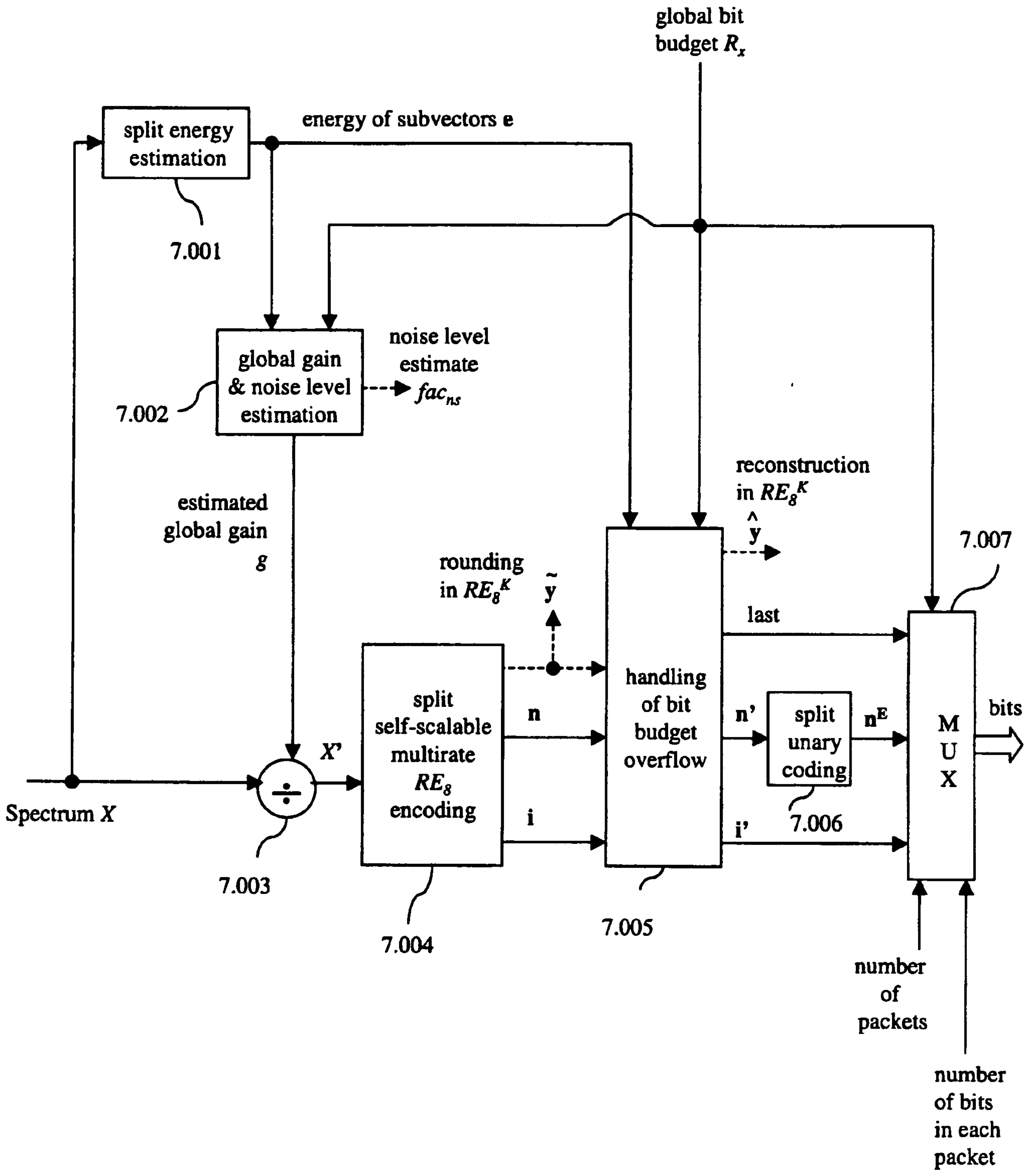
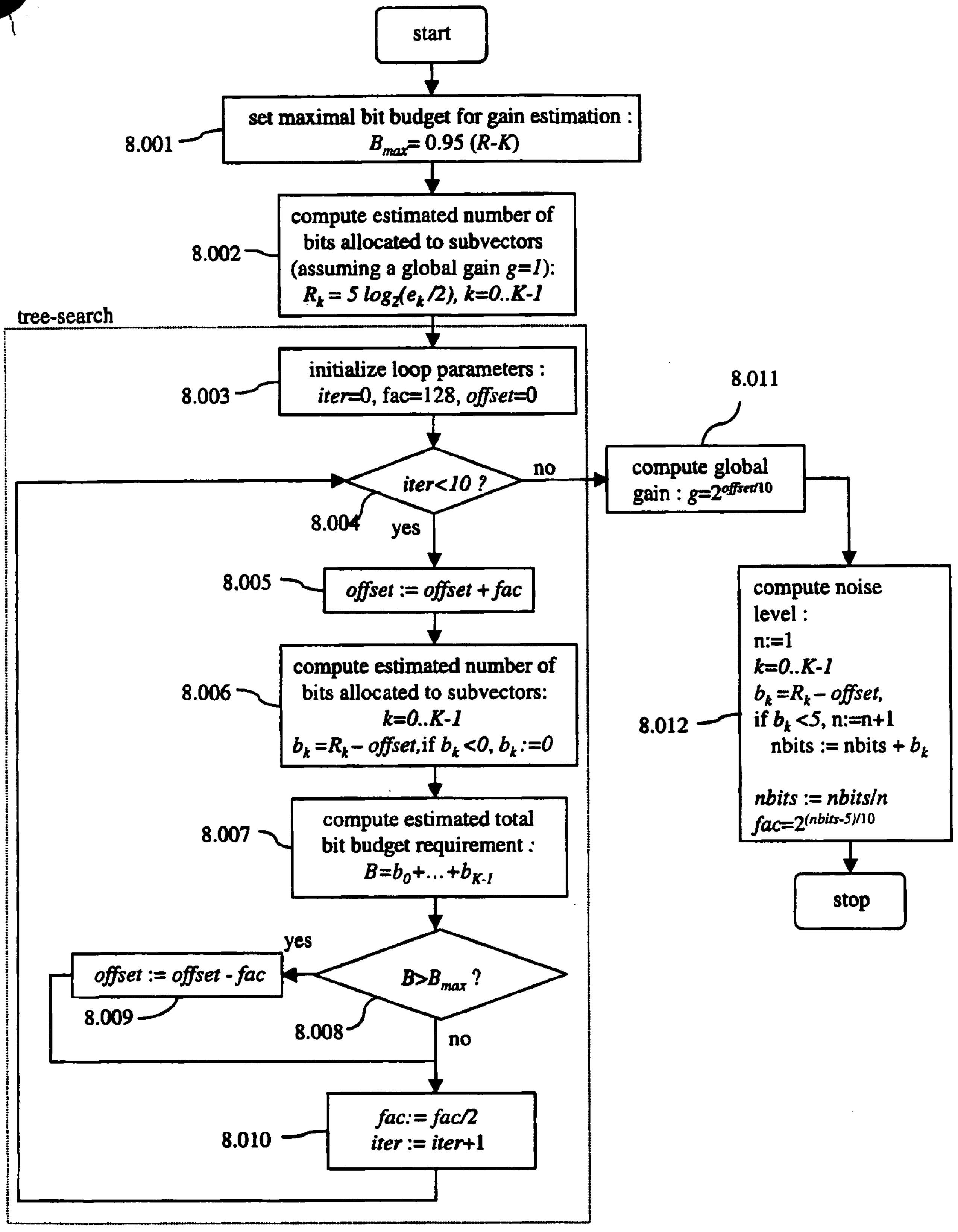

global bit budget R_x
split energy estimation
energy of subvectors e
7.001
global gain & noise level estimation
noise level estimate fac_{ns}
7.002
estimated global gain g
reconstruction in RE_8^K
$\hat{y}$
7.007
rounding in RE_8^K
$\tilde{y}$
last
handling of bit budget overflow
split self-scalable multirate RE_8 encoding
n
n'
split unary coding
n^E
M U X
bits
X'
Spectrum X
i
i'
7.006
7.003
7.004
7.005
number of packets
number of bits in each packet

Fig 7. Algebraic encoding of a set of coefficients based on self-scalable multi-rate RE_8 vector quantization.



start
8.001
set maximal bit budget for gain estimation : B_{max}= 0.95 (R-K)
8.002
compute estimated number of bits allocated to subvectors (assuming a global gain $g=1$): $R_k = 5\log_2(e_k/2)$, $k=0..K-1$
tree-search
8.003
initialize loop parameters : iter=0, fac=128, offset=0
8.004
iter<10 ?
no
yes
8.011
compute global gain : $g=2^{offset/10}$
8.005
offset := offset + fac
8.006
compute estimated number of bits allocated to subvectors: $k=0..K-1$ $b_k = R_k - offset$, if $b_k<0$, $b_k:=0$
8.012
compute noise level : n:=1 $k=0..K-1$ $b_k = R_k - offset$, if $b_k<5$, n:=n+1 nbits := nbits + b_k nbits := nbits/n $fac=2^{(nbits-5)/10}$
8.007
compute estimated total bit budget requirement : $B=b_0+\ldots+b_{K-1}$
stop
yes
offset := offset - fac
$B>B_{max}$?
8.009
8.008
no
8.010
fac:= fac/2 iter := iter+1

Fig 8. Iterative global gain estimation procedure in log-domain.

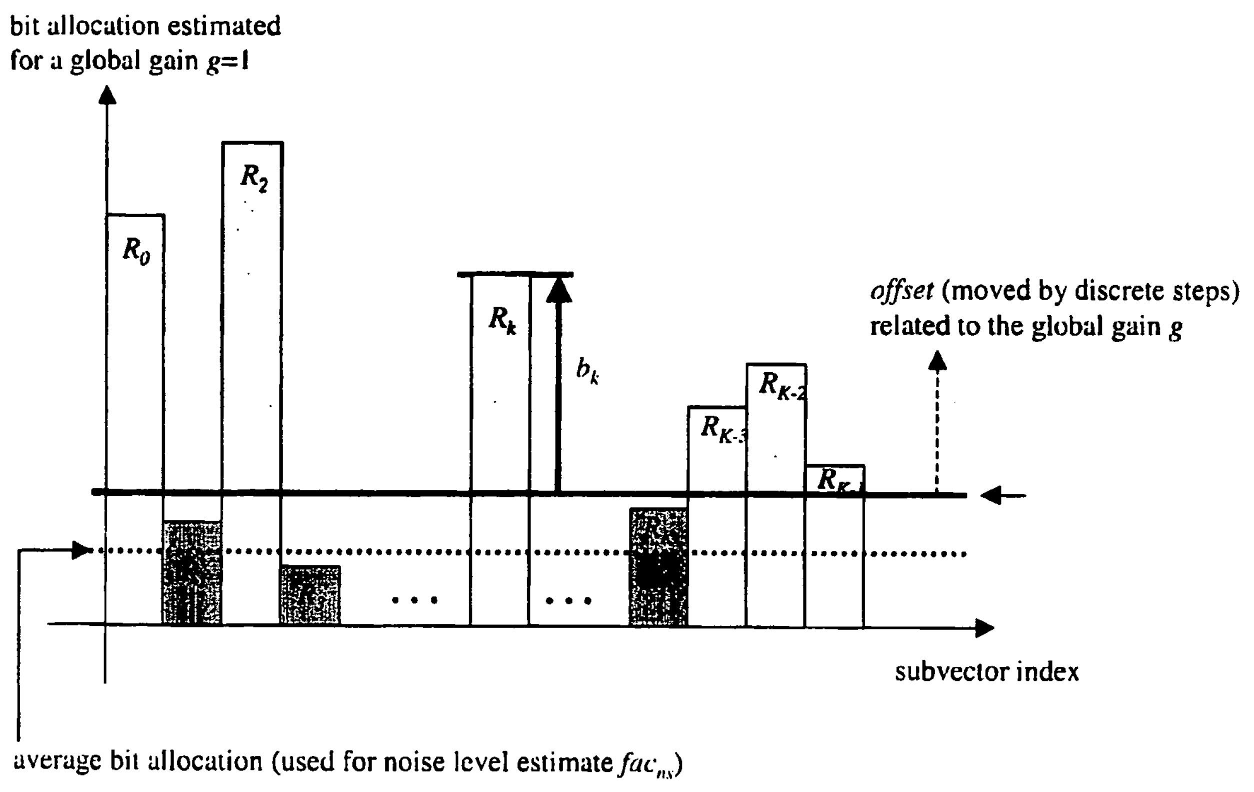

bit allocation estimated
for a global gain g=1
R_0
R_2
R_k
b_k
R_{K-3}
R_{K-2}
R_{K-1}
offset (moved by discrete steps)
related to the global gain g
...
...
subvector index
average bit allocation (used for noise level estimate fac_{ns})

Fig 9. Principle of global gain estimation & noise level estimation (reverse waterfilling).

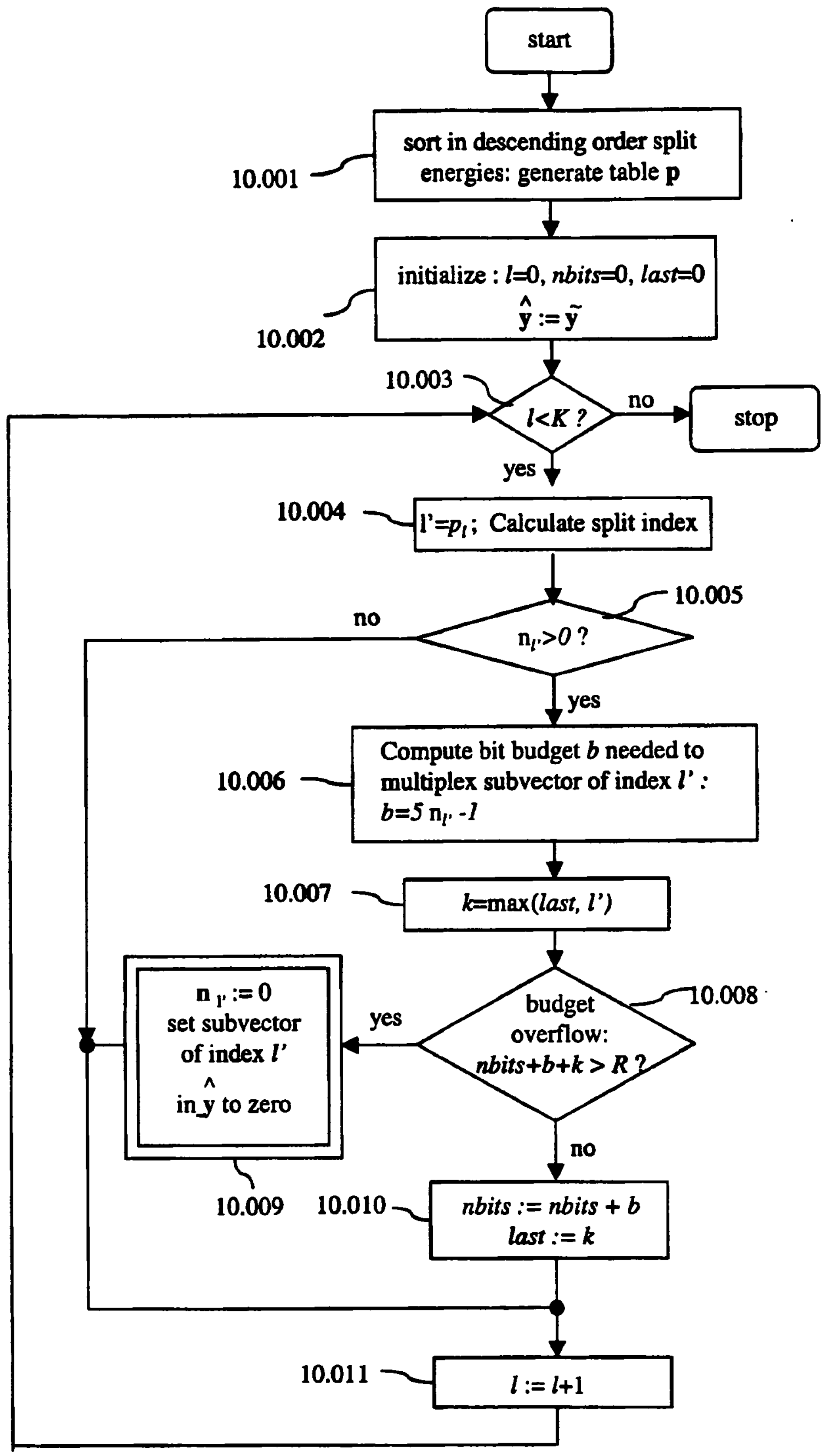


Fig 10. Handling of bit budget overflow.

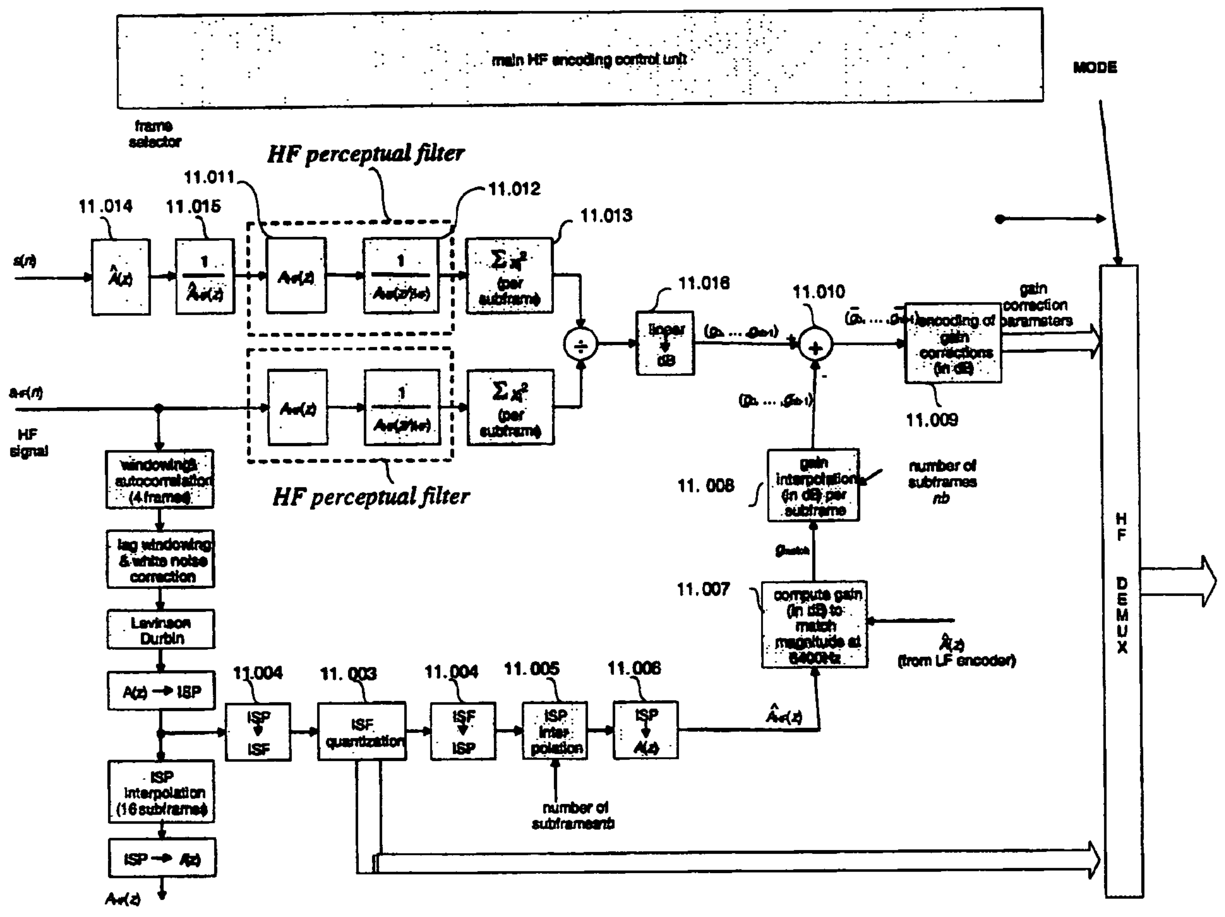


Figure 11. High frequency encoder

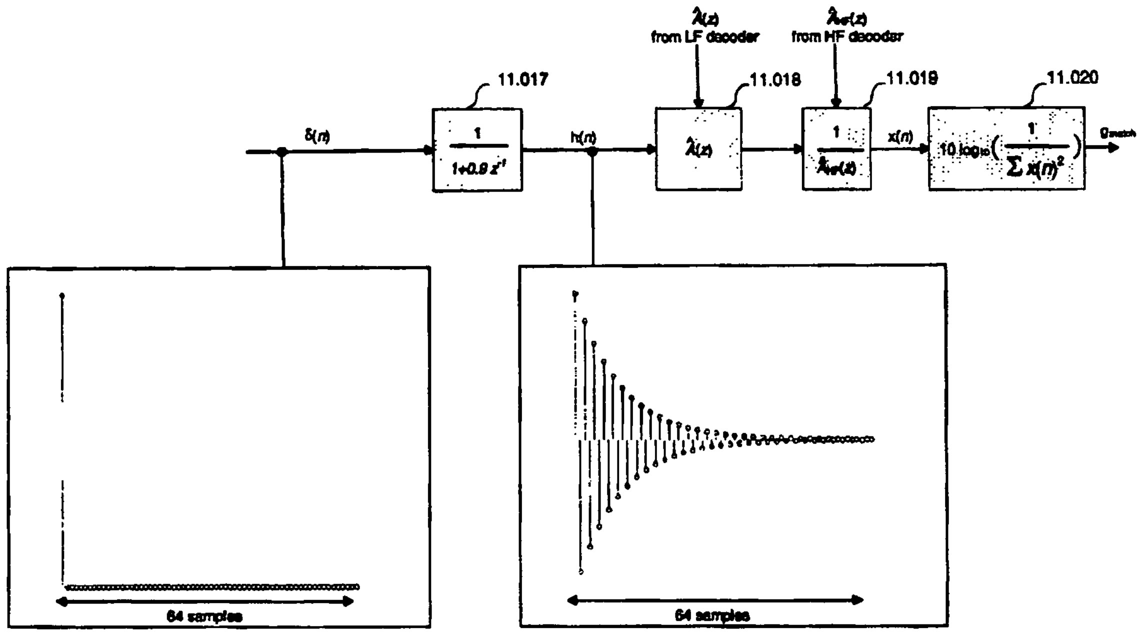


Figure 11a. Gain matching between low and high frequency envelope.

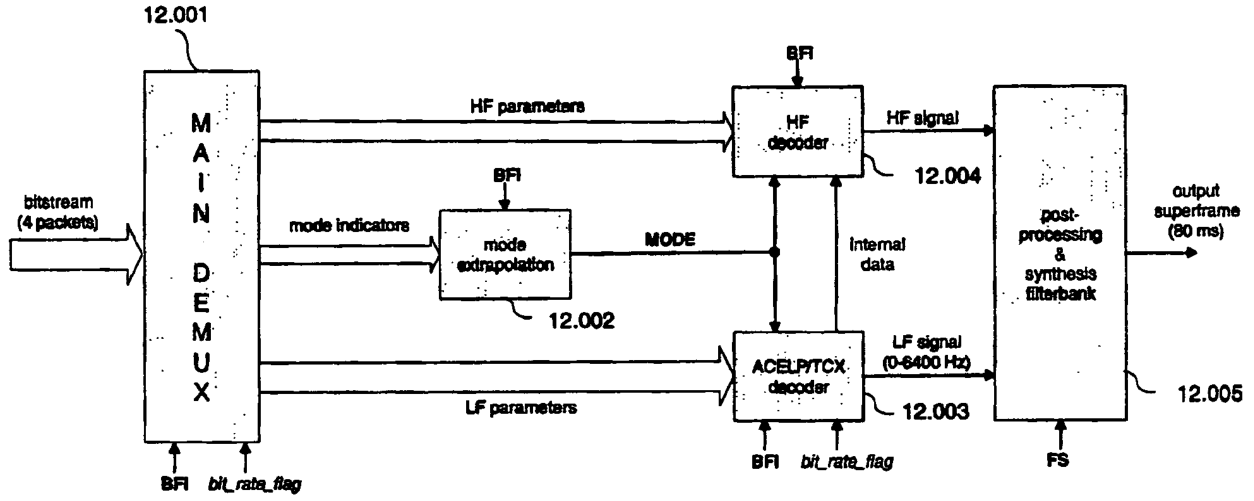
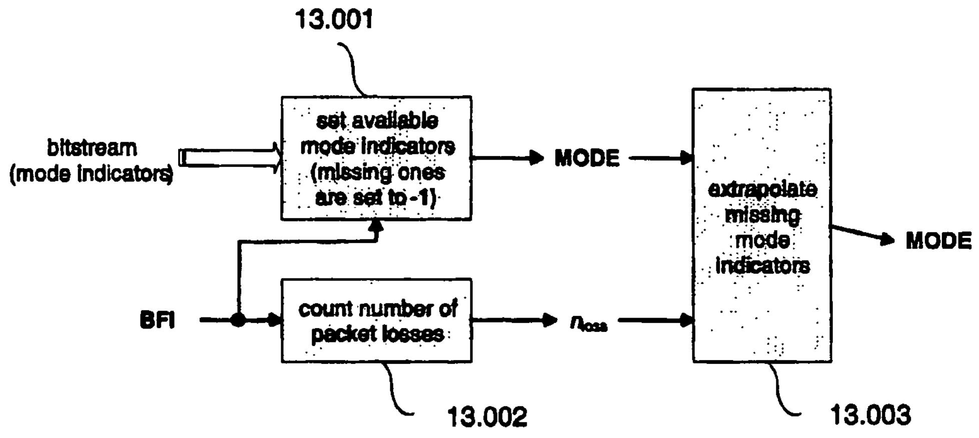


Figure 12 . High level block diagram of the decoder.



13.001
bitstream (mode indicators)
set available mode indicators (missing ones are set to -1)
MODE
extrapolate missing mode indicators
MODE
BFI
count number of packet losses
n_{loss}
13.002
13.003

Figure 13. Mode extrapolation.

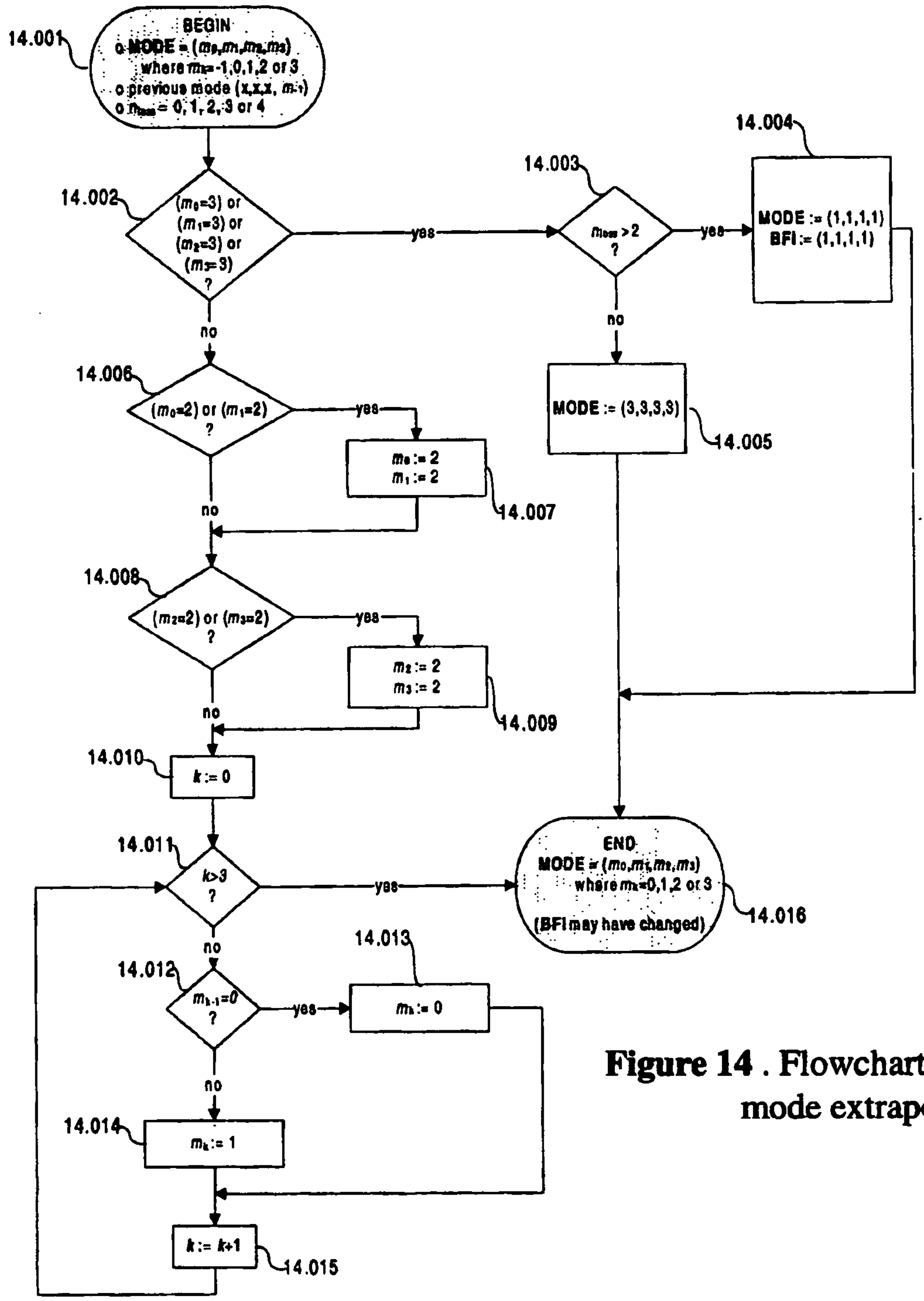


Figure 14 . Flowchart mode extrapolation

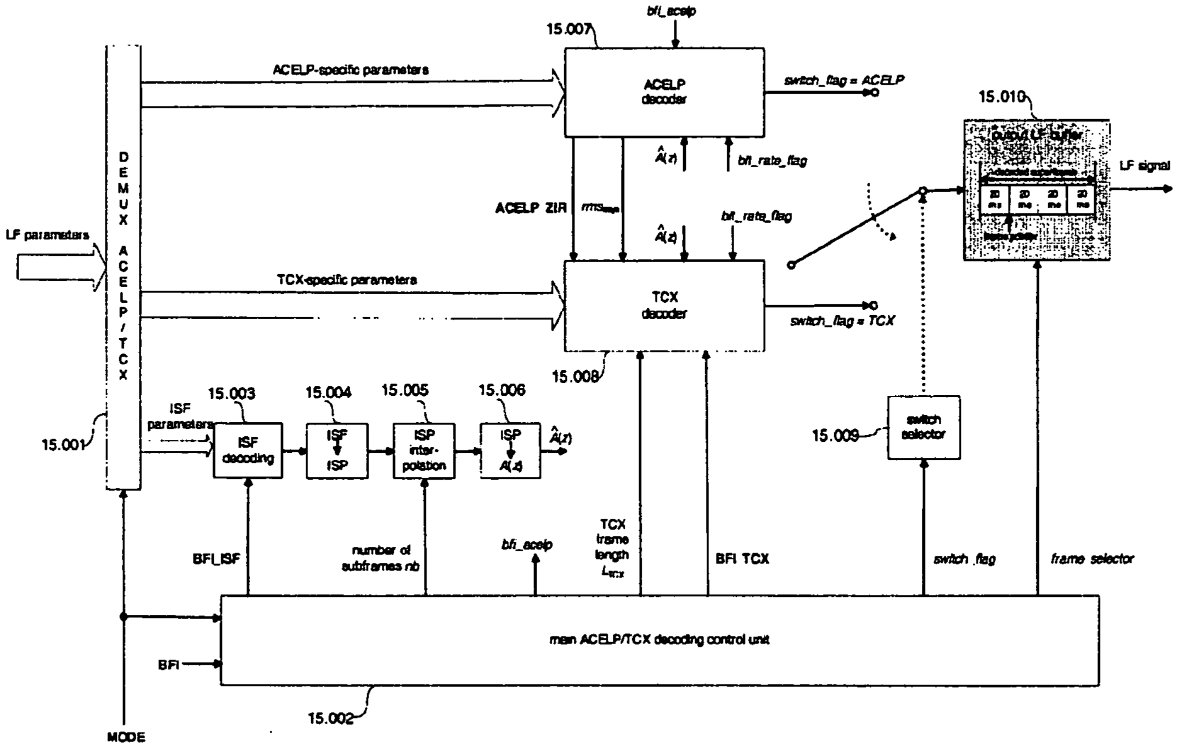


Figure 15. ACELP/TCX decoding.

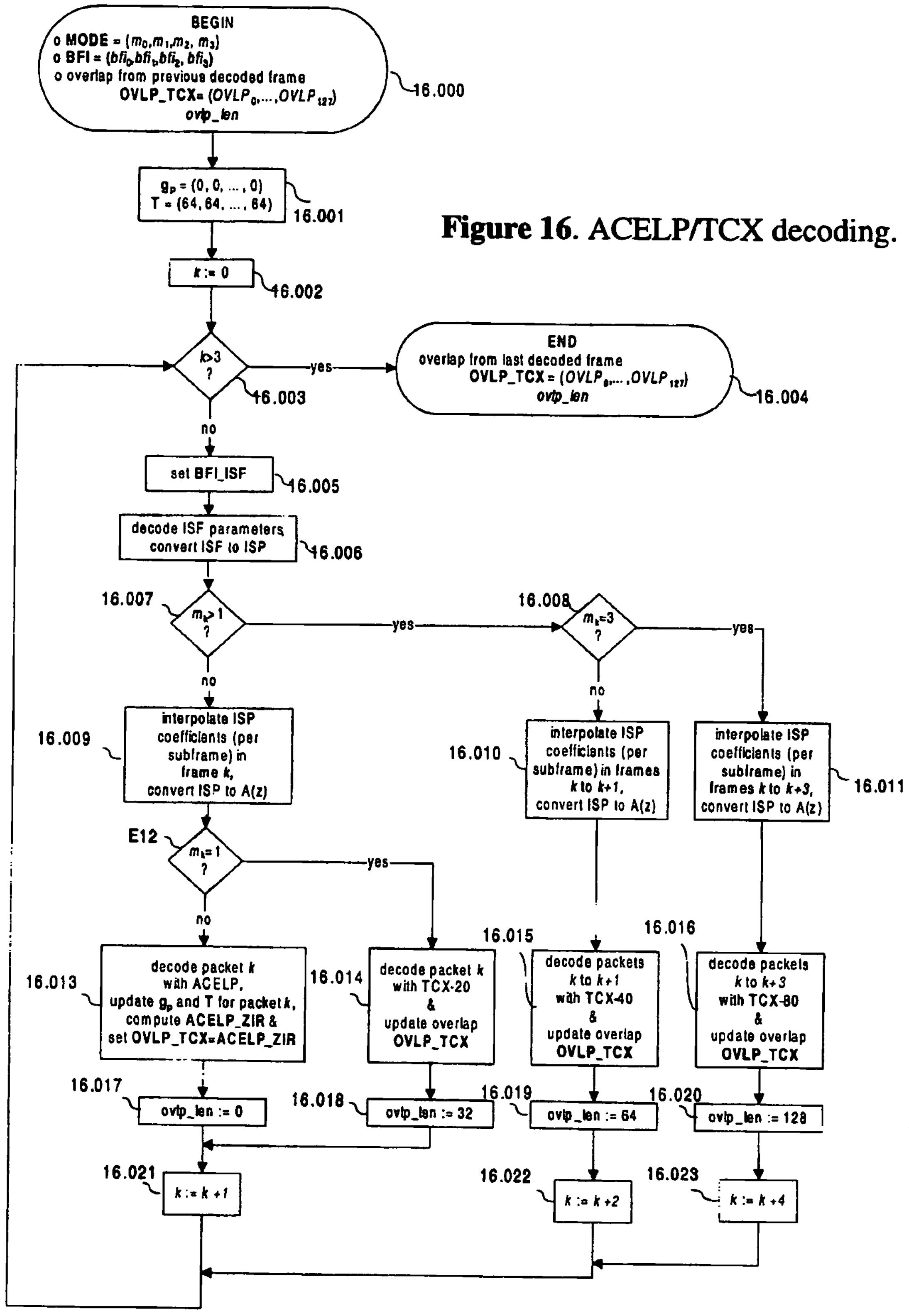
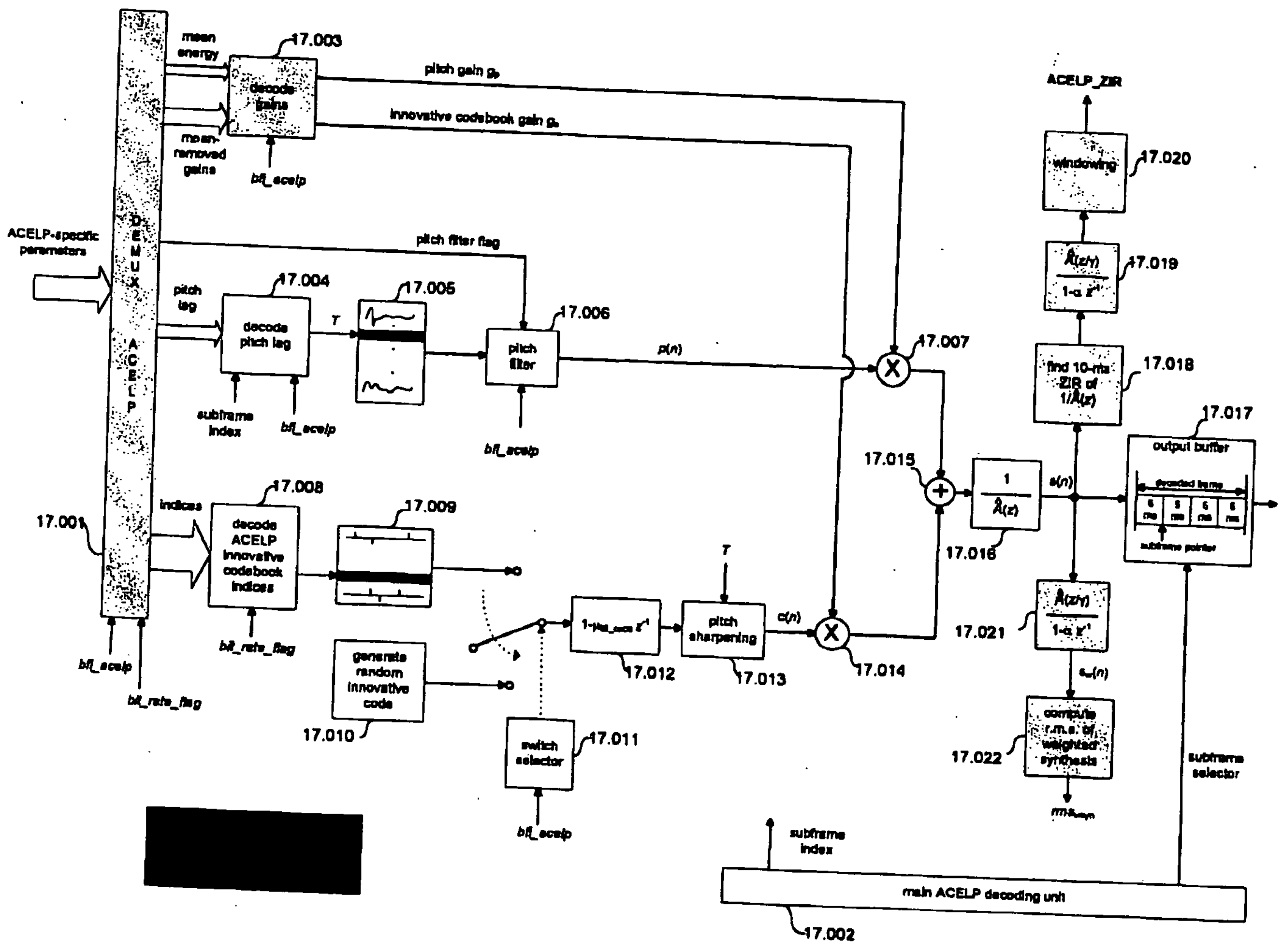

BEGIN
o MODE = (m_0,m_1,m_2, m_3)
o BFI = (bfi_0,bfi_1,bfi_2, bfi_3)
o overlap from previous decoded frame
OVLP_TCX= ($OVLP_0$,...,$OVLP_{127}$)
ovlp_len
16.000
g_p = (0, 0, ..., 0)
T = (64, 64, ..., 64)
16.001
k := 0
16.002
k>3 ?
yes
no
16.003
END
overlap from last decoded frame
OVLP_TCX = ($OVLP_0$,...,$OVLP_{127}$)
ovlp_len
16.004
set BFI_ISF
16.005
decode ISF parameters, convert ISF to ISP
16.006
16.007
m_k>1 ?
16.008
m_k=3 ?
16.009
interpolate ISP coefficients (per subframe) in frame k, convert ISP to A(z)
16.010
interpolate ISP coefficients (per subframe) in frames k to k+1, convert ISP to A(z)
16.011
interpolate ISP coefficients (per subframe) in frames k to k+3, convert ISP to A(z)
E12
m_k=1 ?
16.013
decode packet k with ACELP, update g_p and T for packet k, compute ACELP_ZIR & set OVLP_TCX=ACELP_ZIR
16.014
decode packet k with TCX-20 & update overlap OVLP_TCX
16.015
decode packets k to k+1 with TCX-40 & update overlap OVLP_TCX
16.016
decode packets k to k+3 with TCX-80 & update overlap OVLP_TCX
16.017
ovlp_len := 0
16.018
ovlp_len := 32
16.019
ovlp_len := 64
16.020
ovlp_len := 128
16.021
k := k +1
16.022
k := k +2
16.023
k := k +4

Figure 16. ACELP/TCX decoding.



[redacted]

Figure 17. ACELP decoding.

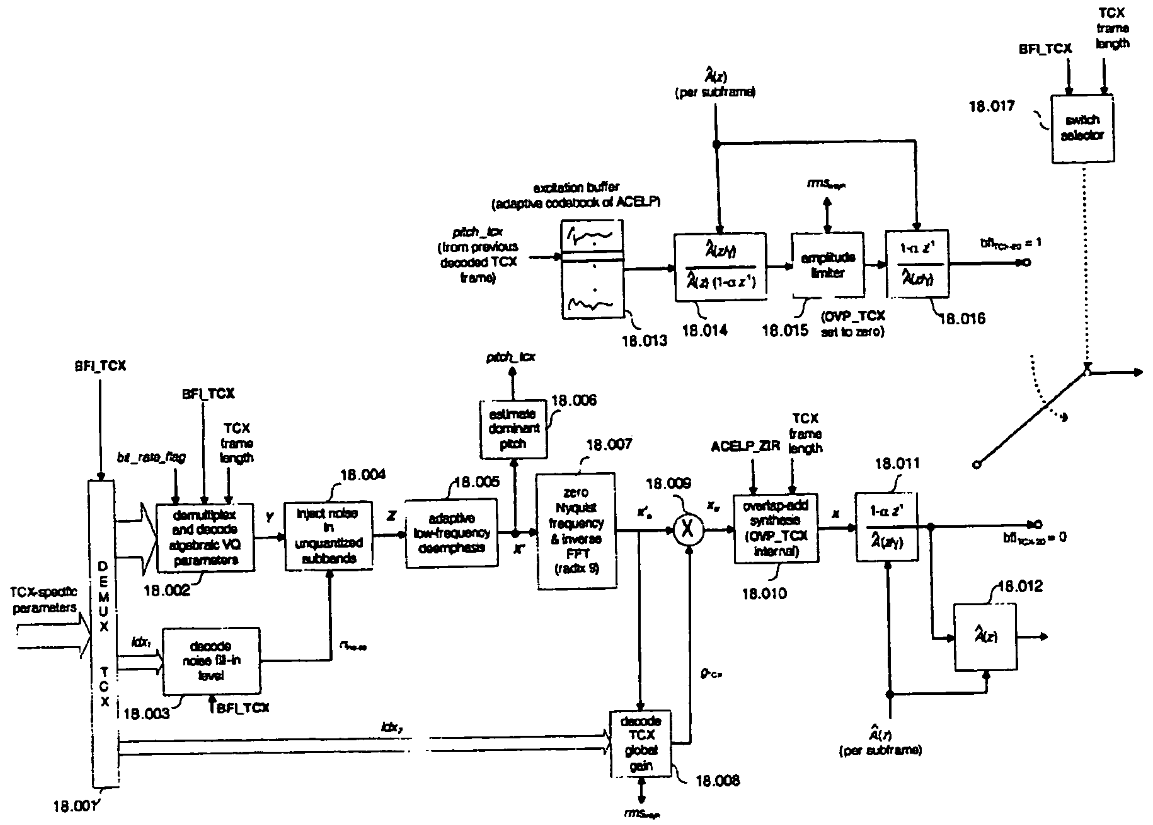

BFI_TCX
TCX frame length
18.017
switch selector
$\hat{A}(z)$ (per subframe)
excitation buffer (adaptive codebook of ACELP)
rms
pitch_tcx (from previous decoded TCX frame)
$\frac{\hat{A}(z/\gamma)}{\hat{A}(z)(1-\alpha z^{-1})}$
amplitude limiter
$\frac{1-\alpha z^{-1}}{\hat{A}(z/\gamma)}$
bfi = 1
18.013
18.014
18.015
(OVP_TCX set to zero)
18.016
BFI_TCX
pitch_tcx
BFI_TCX
18.006
estimate dominant pitch
TCX frame length
bit_rate_flag
18.007
ACELP_ZIR
TCX frame length
18.004
18.005
18.009
18.011
demultiplex and decode algebraic VQ parameters
Y
inject noise in unquantized subbands
Z
adaptive low-frequency deemphasis
X'
zero Nyquist frequency & inverse FFT (radix 9)
x'
X
overlap-add synthesis (OVP_TCX internal)
x
$\frac{1-\alpha z^{-1}}{\hat{A}(z/\gamma)}$
bfi = 0
TCX-specific parameters
D E M U X T C X
18.002
18.012
idx
decode noise fill-in level
n
g
$\hat{A}(z)$
18.010
18.003
BFI_TCX
idx
decode TCX global gain
18.008
$\hat{A}(z)$ (per subframe)
18.001
rms

Figure 18. TCX decoding.

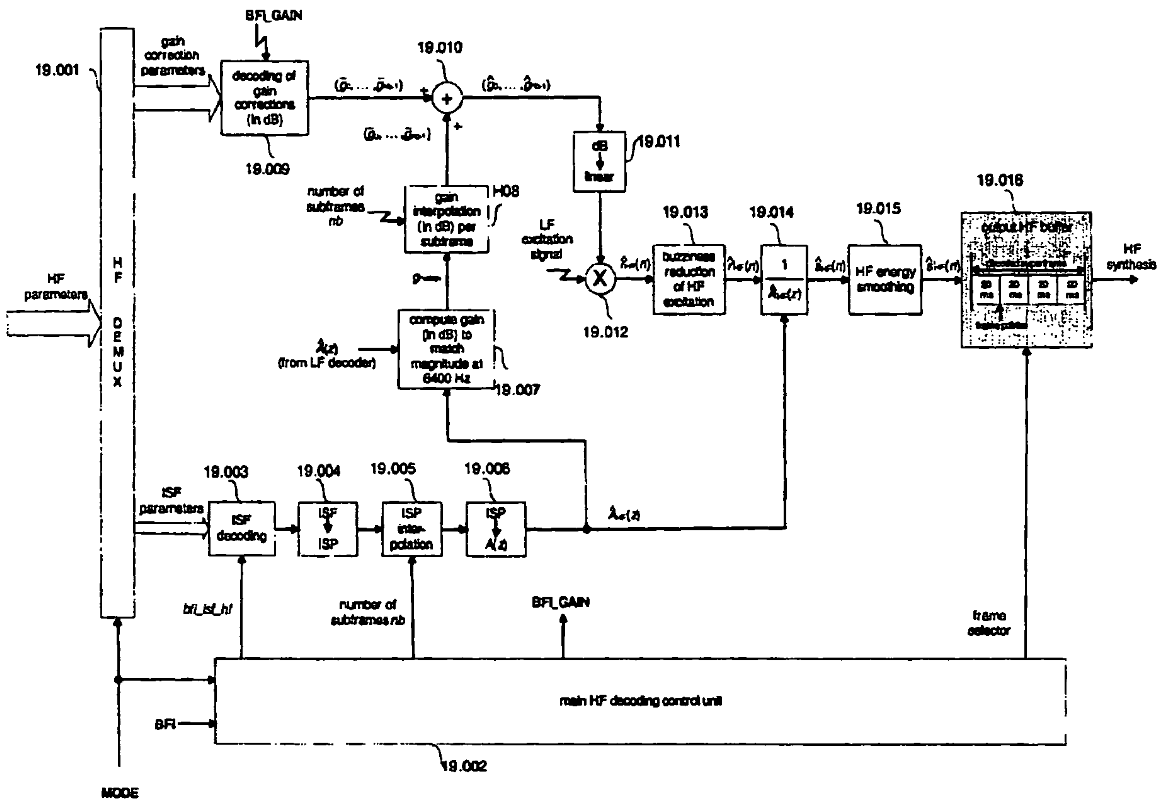


Figure 19. High-frequency decoder.

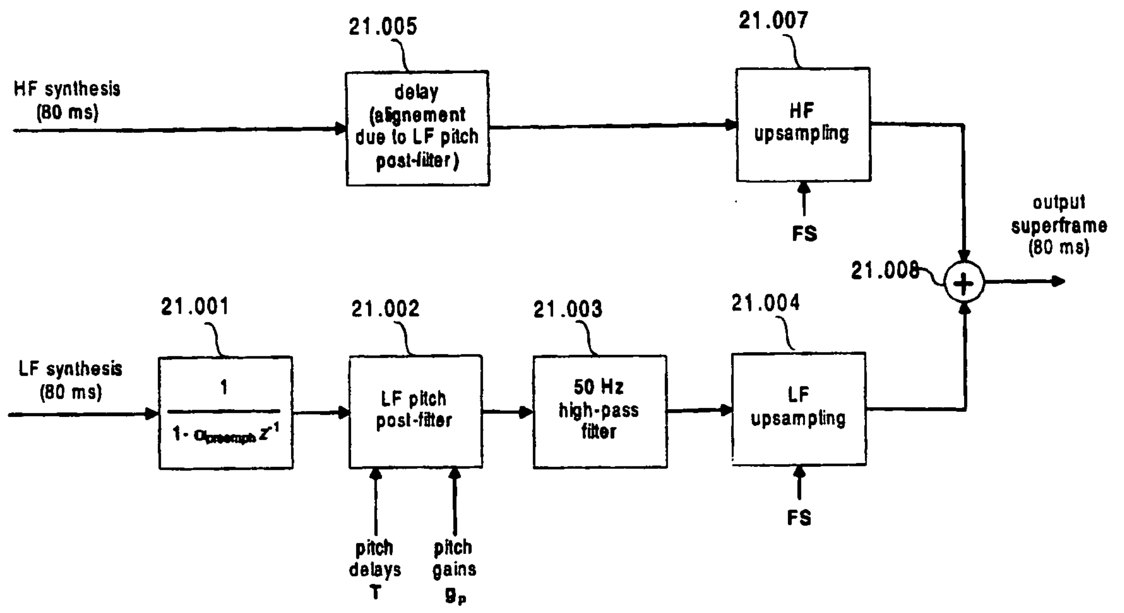


Figure 21. Post-processing and synthesis filterbank.

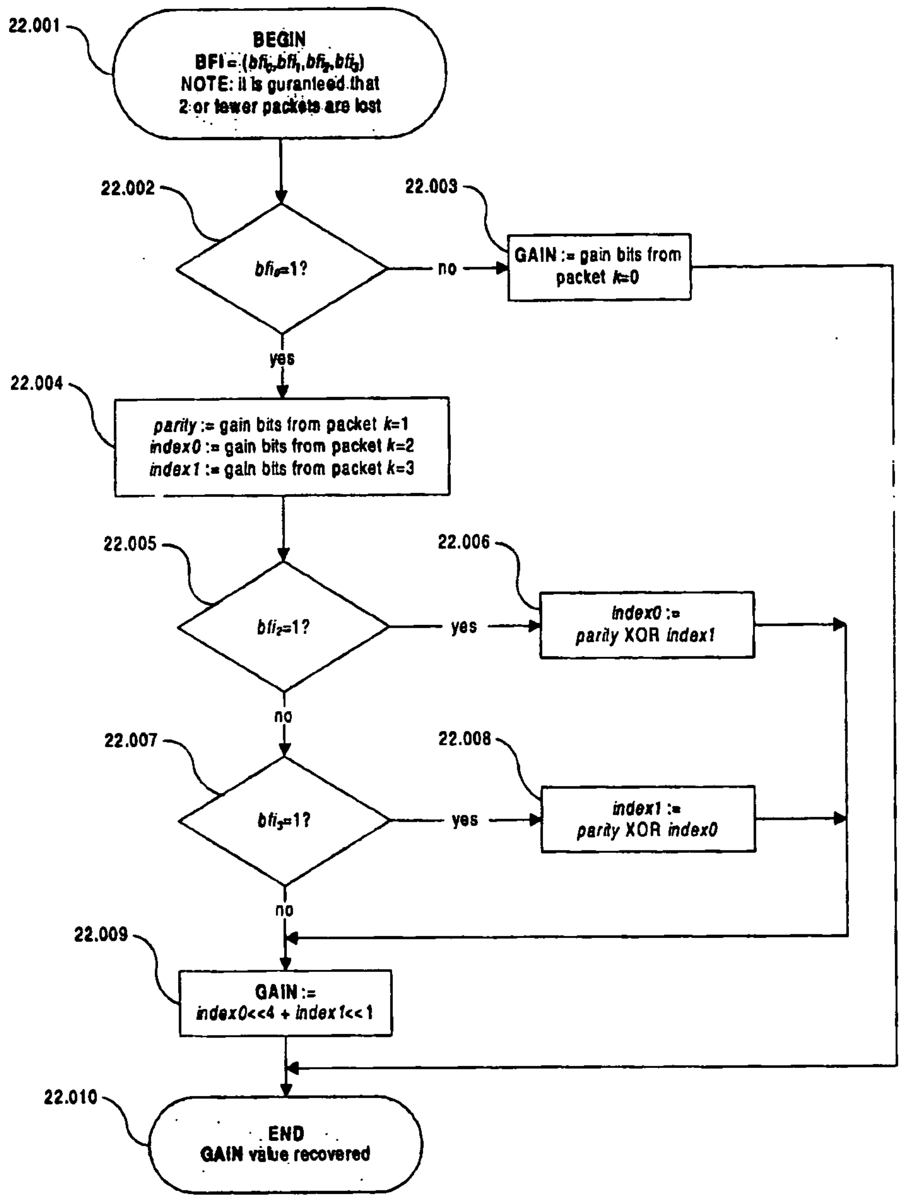


Figure 22. TCX gain decoding.

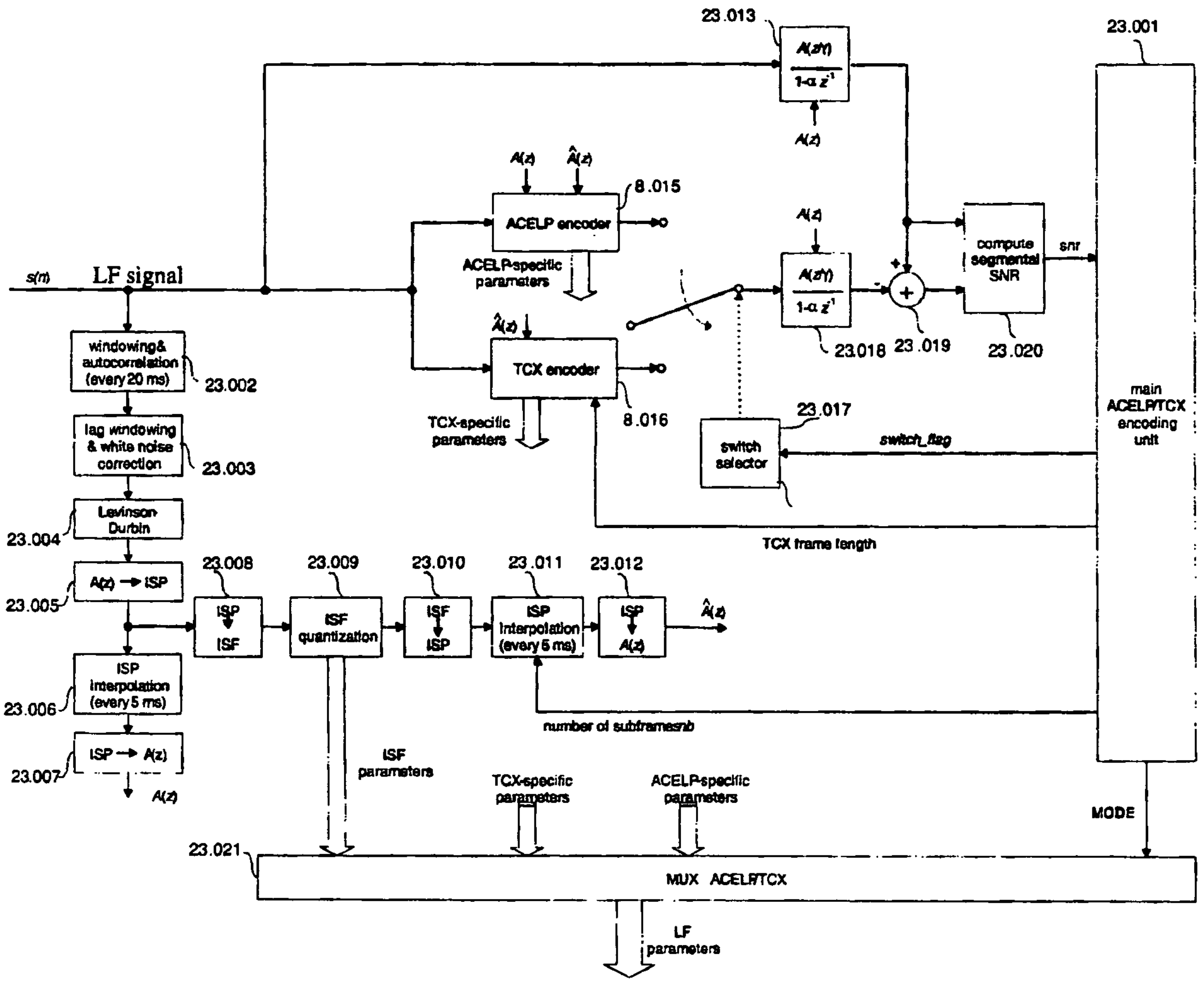


Figure 23. Block diagram of the LF encoder

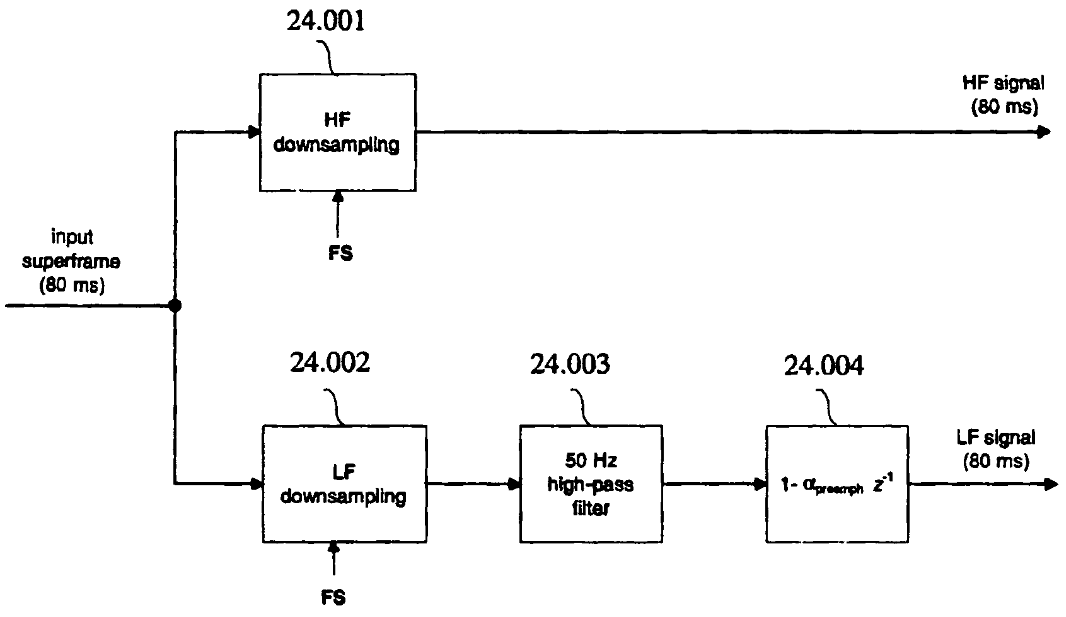


Figure 24. Pre-processing and sub-band decomposition.

Figure 20. Gain computation
In HF decoder (same as Figure 11a)